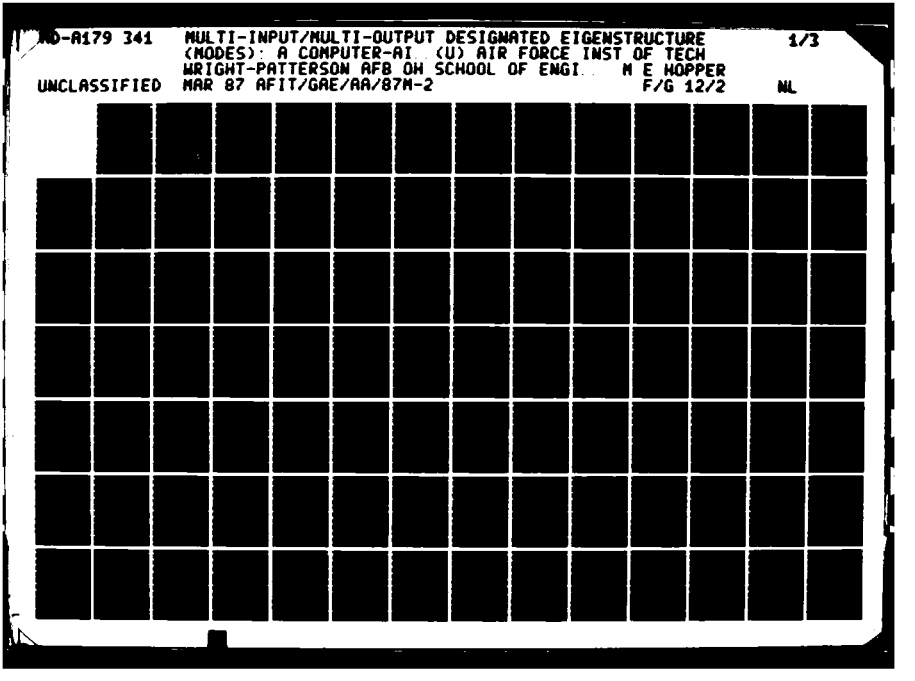
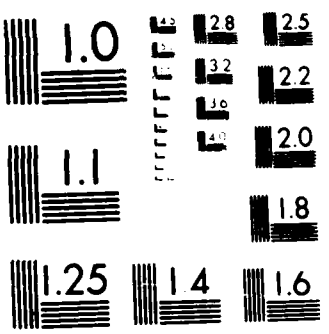
AD-A179 341 MULTI-INPUT/MULTI-OUTPUT DESIGNATED EIGENSTRUCTURE (MODES): A COMPUTER-AI (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI M E HOPPER

1/3

UNCLASSIFIED MAR 87 AFIT/GAE/AA/87M-2 F/G 12/2 NL





MULTI-INPUT/MULTI-OUTPUT DESIGNATED IGENSTRUCTURE (MODES): A COMPUTER-AIDED CONTROL SYSTEM DESIGN PROGRAM

THESIS

Michael E. Hopper
Captain, USAF

AFIT/GAE/AA/87M-2

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

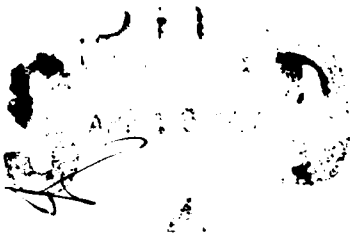
Wright-Patterson Air Force Base, Ohio

MULTI-INPUT/MULTI-OUTPUT DESIGNATED EIGENSTRUCTURE (MODES): A COMPUTER-AIDED CONTROL SYSTEM DESIGN PROGRAM

THESIS

Michael E. Hopper
Captain, USAF

AFIT/GAE/AA/87M-2



AFIT/GAE/AA/87M-2

MULTI-INPUT/MULTI-OUTPUT DESIGNATED EIGENSTRUCTURE (MODES): A COMPUTER-AIDED CONTROL SYSTEM DESIGN PROGRAM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Aeronautical Engineering

Michael E. Hopper, B.S.
Captain, USAF

March 1987

Acknowledgments

In developing this program and writing this thesis I received a lot of support and help. Many thanks go to my faculty advisor, Capt. Dan Gleason, for his suggestions which have made MODES a "well-rounded" program. Thanks also go to Capt. Tom Barth, a fellow student who has been a great help by expanding the boundaries of my computer knowledge. To my boss, Maj. Bohdan Kunciw, a word of thanks, for without his support, I never would have been able to take all of my graduate courses or complete this thesis as a part-time student. And finally to my wife Tina, a special thank you for your support, both moral and typing.

Table of Contents

Notation

A, $A(t)$	open-loop system matrix
$\tilde{A}$	closed-loop, modified system matrix
A^T	transpose of the matrix A
A^{-1}	inverse of the matrix A
A^+	pseudoinverse of the matrix A
A_T	discrete representation of e^{AT}
a_{ij}	element in i^{th} row and j^{th} column of the matrix A
B, $B(t)$	input distribution matrix
B_T	discrete representation of $\left(\int_0^T e^{A\xi} d\xi \right) B$
G, $G(t)$	feedback gain matrix
I	identity matrix
P	matrix of eigenvectors
P_o	matrix of optimal eigenvectors
R	matrix product GP_o
r	rank of a matrix
S	matrix of non-zero singular values
U	matrix of left singular vectors of a matrix A; also, the matrix of orthonormal eigenvectors of AA^T
$\bar{u}_i$	i^{th} column of U
$\bar{u}$, $\bar{u}(t)$	input vector
V	matrix of right singular vectors of a matrix A; also, the matrix of orthonormal eigenvectors of A^TA
$\bar{v}_i$	i^{th} column of V
$\bar{x}$, $\bar{x}(t)$	state vector

Λ	diagonal matrix of eigenvalues
λ_i	i^{th} eigenvalue
Σ	matrix containing the matrix S in the first r rows and columns, and zero-valued elements elsewhere
σ_i	i^{th} singular value
$\|a\|$	absolute value of the scalar a
$\|\|\bar{v}\|\|$	magnitude of the vector $\bar{v}$
$\|\|A_F\|\|$	Froebinius norm of the matrix A
$\|\|A\|\|_\infty$	uniform norm of the matrix A

Abstract

As a minimum, in the design of control systems, the engineer wants the controlled system to exhibit certain acceptable response characteristics which approach an ideal. Specifying the closed-loop response of a system, including both the modes and mode shapes, can be accomplished with the eigenstructure assignment approach to system synthesis. This effort was to create an interactive computer-aided control system design program to enable the designer to use the modern control approach to design a control system with specified response characteristics, or to come as close to the ideal as possible. With this eigenstructure assignment program, the design engineer can easily calculate the feedback gains needed to give the controlled system the optimum response characteristics by specifying the desired eigenvalues and eigenvectors.

Given a linear, time-invariant state-space model of a controlled system with a system matrix A and an input distribution matrix B, the program MODES uses the singular value decomposition to find the range space of achievable eigenvectors of the augmented matrix $[A - \lambda_i I, B]$. Here, λ_i is a desired eigenvalue. The singular value decomposition is used again to project the desired eigenvector $\bar{p}_{di}$ into the range space of achievable eigenvectors. This projection yields the eigenvector which comes as close to matching the desired eigenvector as is possible. This projection is termed the optimal eigenvector, $\bar{p}_{oi}$, associated with the desired eigenvalue λ_i. The optimal eigenvectors are collected into a matrix of eigenvectors, P_o. Because of the

form of the augmented matrix $[A - \lambda_i I, B]$, when we find the matrix of optimal eigenvectors, we also can find the matrix product GP_o where G is the feedback gain matrix required to give the controlled system the optimal eigenstructure. To solve for the feedback gain matrix G, we only have to postmultiply the postmultiply the product GP_o by the inverse of P_o, P_o^{-1}.

The program MODES performs these calculations. In addition, MODES can provide valuable information about the controlled system to assist the designer in specifying the desired eigenstructure. By calculating the eigenvalues and eigenvectors of the original system, by plotting the time response history of the original system, and by calculating the resolvent matrix, poles and zeroes for classical control analysis, MODES is a strong tool in determining how the eigenstructure should be altered. By presenting the same information for the modified "optimized" system, MODES can show where additional changes to the specified desired eigenstructure should be made.

The accuracy of MODES is verified with sample applications found in the technical literature.

MULTI-INPUT/MULTI-OUTPUT DESIGNATED EIGENSTRUCTURE (MODES): A COMPUTER-AIDED CONTROL SYSTEM DESIGN PROGRAM

I. Introduction

With the wide variety of computer resources available in industry today, ranging from the desk top micro-computer to the "super computer", the control system design engineer is no longer constrained to perform the many required design algorithms by hand. Nor is he required to write and test new computer algorithms for each new set of design constraints. Today, there is a growing number of sophisticated computer-aided control system design (CACSD) packages for control system development. CACSD technology has greatly enhanced productivity and will continue to do so as more powerful packages become available, and as new design routines are added to existing CACSD systems to give them greater versatility (Spang, 1984: 1724).

One control system design tool available to the engineer, but not explicitly found in most CACSD systems, is the state-space approach for system synthesis by state variable feedback known as eigenstructure assignment. For the linear time-invariant model

$$\dot{\bar{x}}(t) = A\bar{x}(t) + B\bar{u}(t)$$

a suitable algorithm can be written which will generate the feedback gain matrix which most closely places the eigenvalues and eigenvectors of the controlled system to their desired design values. Such an algorithm is the subject of this thesis.

As stated by Walker, Shah, and Gupta (Walker, et al, 1984: 1744), one of the goals for computer-aided engineering is to accelerate and encourage generation of new ideas and approaches for computer packages. The design program "Multi-Input, Multi-Output Designated Eigenstructure" (MODES) is an approach which could easily be integrated into existing comprehensive, flexible CACSD packages, or can stand alone as a single-purpose design program to be used by the design engineer in conjunction with other programs. As a stand-alone program, MODES may be most useful in the classroom environment as an aid in the instruction of the eigenstructure approach to control system design.

Overview of MODES.

MODES is an interactive program designed for the first time user as well as the user who has become very familiar with its features. The first message to appear asks the user if he wants a file of the program output saved as a computer file. The second message asks if an introductory message is desired. Next, the user is asked if he wants full output including intermediate results, or if he wants only the information of major interest.

The first information to be entered about the specific problem to be worked by MODES are the matrices A and B describing the original system. These matrices may be entered from the keyboard, or read from existing computer files. Whether the matrices are entered from the keyboard or read from an existing computer file, the user is allowed to edit the matrices and to save them as new computer files. Then MODES asks if the user wants to see the eigenstructure of the basic system. If the user wants, MODES will then perform a discrete time simulation of the

basic system of equations, assuming zero intial conditions, and plot the time response characteristics on the terminal screen. MODES then offers to print the resolvent matrix, poles and zeroes of the system. This is the point at which the user is prompted to enter the desired eigenvalues and eigenvectors. As with the system matrices, the desired eigenstructure can be entered from the keyboard or read from an existing computer file. Also, the user may edit the eigenstructure matrices whether they were entered from the keyboard or read from an existing computer file. The eigenstructure matrices may be saved for future use. Because complex eigenvalues must appear with their complex conjugate, when the user enters the eigenstructure from the keyboard, MODES will automatically enter the conjugate to every complex number.

At this time, MODES performs a singular value decomposition on the system, as described in chapter II, to find the feedback gain matrix G which most closely alters the system eigenstructure to that specified by the user. After displaying this feedback gain matrix, MODES defines the matrix $\tilde{A}$ such that

$$\tilde{A} = (A + BG) = P\Lambda P^{-1}$$

where the columns of P are the achievable eigenvectors, and Λ represents the diagonal matrix of desired eigenvalues (see chapter II for more details). The user has the option of saving both the feedback gain matrix and the final modified $\tilde{A}$ matrix. Next, MODES offers to print the actual eigenstructure of the final modified system as well as the original system for convenient comparisons.

Now MODES asks if the user wants a discrete time

simulation of the modified system plotted on the terminal screen. Finally, MODES offers to calculate the resolvent matrix and the poles and zeroes of the modified $\tilde{A}$ matrix. This information can be useful when examining the system using frequency domain techniques.

Hardware Used for Thesis Development.

This thesis was developed with the computer resources of the Air Force Institute of Technology (AFIT) as well as my own equipment. The primary computer resource used was the AFIT VAX/VMS Classroom Support Computer (CSC) with its resident Fortran compiler. My personal equipment consists of an Epson Equity I computer (IBM PC compatible) with a Hayes-compatible modem and Panasonic KX-P1091 printer. In addition to running on the VAX/VMS, MODES has been compiled for both the IBM PC compatible computers as well as the Atari 520ST home computer. For the IBM PCs, version 3.2 of the Microsoft FORTRAN77 compiler was used; For the Atari 520ST, the Pro Fortran-77 compiler by Prospero Software was used. For both of these home computer compilers, the maximum size of the system's A matrix was reduced from 15 by 15 to 10 by 10. Even so, the IBM PC executible program exceeds 220,000 bytes in size, and cannot run on a machine with 256K RAM or less. To date, the PC version of MODES has run on the Zenith Z-150, Z-241 and Z-248 computers, as well as the Columbia and Victor computers. Because of its excellent error messages, I used the WATFOR-77 Fortran interpreter by WATCOM Systems, Inc. Although memory limitations does not allow the WATCOM interpreter to run the complete MODES program, I was able to make great use of the interpreter when conducting the "bottom-up" testing of each module.

Summary.

MODES is an interactive computer-aided design program ideally suited for classroom instruction of the eigenstructure assignment approach to control system design. By showing the designer the eigenstructure of the equations describing the basic system and plotting the time response of that system and calculating the system poles and zeroes, MODES enables the designer to see better how the eigenstructure should be modified to give it the desired characteristics. Once MODES has determined the optimal feedback gain matrix, the closed-loop eigenstructure, the closed-loop time response characteristics and the closed-loop poles and zeroes are plotted on the screen.

With the modular approach taken during the development of MODES, one should have little difficulty in appending it to an existing package of other control system design programs, or modifying it to meet new needs.

Thesis Overview.

The remainder of this thesis is divided into five sections, Chapters II through VI, along with two appendices.

Chapter II discusses the theory of the eigenstructure assignment approach to control system design. The theory of singular value decomposition is presented, as well as its application in MODES. All essential steps in the eigenstructure assignment problem are presented.

Chapter III presents two sample cases of this design approach found in the technical literature, and uses them to verify the correctness of the MODES coding. The first case is based on the AFTI/F-16 demonstrator aircraft's pitch-pointing features. The second case is a look at a flight-

test maneuver autopilot for the F-15 fighter aircraft.

Chapter IV presents a discussion of the major subroutines found in MODES. The subroutines for performing the singular value decomposition, setting up the matrices for input to the singular value decomposition routine, calculating the eigenstructure of a general matrix, defining and printing the eigenstructure, calculating the modified system matrix, discretizing the system response, plotting the system response, finding the resolvent matrix and the poles and zeroes of the resolvent matrix, calculating the inverse of a square matrix and performing common utilities are presented.

Chapter V is the users' guide for MODES. The conclusions are found in Chapter VI. The appendices contain the coding for MODES. Appendix A lists the main program, and Appendix B lists all of the subroutines.

II. Approach to Eigenstructure Assignment (Elbert, 1984: 148-174)

The general linear system is described by the state equation:

$$\dot{\bar{x}}(t) = A(t)\bar{x}(t) + B(t)\bar{u}(t)$$

For the open-loop control system, the designer must find a control function $\bar{u}(t)$ for the time interval of interest such that the state vector $\bar{x}(t)$ varies in a desired manner. The control function is dependent on time only. For the closed-loop control system, the control function is dependent on the states and perhaps on time. Therefore, for the closed-loop system, the state equation can be written as

$$\dot{\bar{x}}(t) = A(t)\bar{x}(t) + B(t)\bar{u}(\bar{x}(t),t)$$

If we assume a linear controller, the control function can be expressed as

$$\bar{u}(\bar{x}(t),t) = G(t)\bar{x}(t)$$

where $G(t)$ is the feedback gain matrix.

Substituting this into the closed-loop state equation gives

$$\dot{\bar{x}}(t) = A(t)\bar{x}(t) + B(t)G(t)\bar{x}(t)$$

$$= [A(t) + B(t)G(t)]\bar{x}(t)$$

When $A(t)$, $B(t)$ and $G(t)$ are not time dependent, we can

write our state equations as

$$\dot{\bar{x}}(t) = [A + BG]\bar{x}(t)$$

For notational convenience, denote

$$\tilde{A} = A + BG \tag{2.1}$$

This gives

$$\dot{\bar{x}}(t) = \tilde{A}\bar{x}(t)$$

Therefore, the response characteristics of the linear, time-invariant, closed-loop state variable feedback control system are determined by the eigenstructure of the matrix $\tilde{A}$. The design synthesis implemented by MODES consists of calculating the feedback gain matrix G such that the eigenstructure of the $\tilde{A}$ matrix produces the desired system response characteristics.

When specifying the eigenstructure of the $\tilde{A}$ matrix, and therefore the response characteristics of the controlled system, the diagonalizing matrix of eigenvectors, P, and the diagonal matrix of eigenvalues, Λ, are defined. Here, the columns of P are the eigenvectors of $\tilde{A}$, and Λ is given as

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}$$

where λ_1 is the eigenvalue associated with the eigenvector found in the first column of P, λ_2 is the eigenvalue associated with the eigenvector found in the second column of P,

and so on. This gives

$$\tilde{A} = P\Lambda P^{-1} \qquad (2.2)$$

For the special case when the input distribution matrix B has the same dimensions as A, and B is non-singular, we can find a unique feedback gain matrix G as follows. Combining Eqs (2.1) and (2.2) yields

$$BG = \tilde{A} - A$$

$$= P\Lambda P^{-1} - A$$

which gives

$$G = B^{-1}[P\Lambda P^{-1} - A]$$

Because practical systems do not often occur such that the number of inputs equals the number of states, this expression is of limited use. In most cases, the input distribution matrix B is not square, which leaves either no solutions for G, a single unique solution, or many solutions. We must now explore if one or more solutions exist.

Since the columns of P from Eq (2.2) are the eigenvectors $\bar{p}_i$ associated with eigenvalue λ_i, $\bar{p}_i$ and λ_i must satisfy the defining relation for an eigenvector

$$\tilde{A}\bar{p}_i = \lambda_i \bar{p}_i$$

Substituting Eq (2.1) gives

$$(A + BG)\bar{p}_i = \lambda_i \bar{p}_i = \lambda_i I \bar{p}_i$$

where I is the identity matrix. This gives

$$A\bar{p}_i + BG\bar{p}_i - \lambda_i I\bar{p}_i = 0$$

or

$$[A - \lambda_i I]\bar{p}_i + BG\bar{p}_i = 0$$

Because $\bar{p}_i$ is a column vector, the product $G\bar{p}_i$ is also a column vector. This allows us to partition this expression as

$$[A - \lambda_i I \ , \ B] \begin{bmatrix} \bar{p}_i \\ \hline G\bar{p}_i \end{bmatrix} = 0$$

We now have an expression which establishes the possible values of the eigenvector $\bar{p}_i$ as well as the feedback gain matrix G relative to an eigenvalue λ_i.

When A is dimensioned n by n and B is n by m, the number of inputs available is m. This means that the (n+m) dimensioned vector

$$\begin{bmatrix} \bar{p}_i \\ \hline G\bar{p}_i \end{bmatrix} \qquad (2.3)$$

must lie in the null space of the n by (n+m) matrix:

$$[A - \lambda_i I \ , \ B] \qquad (2.4)$$

Therefore, we must find the null space of the augmented

matrix in expression (2.4), and then select the optimal vector in the form of expression (2.3) from that null space which gives a mode shape for the mode corresponding to the eigenvalue λ_i which comes as close to matching the desired mode shape as is possible. This optimal vector is found by projecting the desired eigenvector into the first n rows of the null space of expression (2.4). As will be shown in the last section of this chapter, the null space of a matrix can be found by performing the singular value decomposition of the matrix. The dimensions of the null space matrix for the matrix in expression (2.4) are (n+m) by (n+m-r), where r is the rank of expression (2.4). We repeat this procedure for each eigenvalue λ_i and eigenvector $\bar{p}_i$. From the lower partitions of the n optimal vectors of the form in expression (2.3), we form the m by n matrix R such that

$$R = [G\bar{p}_{o1} , G\bar{p}_{o2} , \ldots , G\bar{p}_{on}]$$

where the subscript o refers to the optimal eigenvector. This is equivalent to

$$R = GP_o$$

Since the diagonalizing matrix of eigenvectors P_o is nonsingular, we can easily solve for the required feedback gain matrix G,

$$G = RP_o^{-1} \qquad (2.5)$$

Therefore, once a permissable set of eigenvectors is specified, we have a unique feedback gain matrix, G. The only limitation for this procedure is that the calculated

optimal eigenvectors must be linearly independent so that P_o is nonsingular, and therefore invertible.

Selection of Optimal Eigenvectors from a Subspace.

Because the control designer has specified a particular eigenstructure, and wants the controlled system's eigenstructure to be as close to the desired eigenstructure as possible, some method must be used to select the "best" or optimal eigenvector from the null space of the matrix given in expression (2.4). In other words, the null space of expression (2.4) defines the range space of achievable system eigenvectors. From this range space, the eigenvector which most nearly matches the desired eigenvector (that is, the optimal eigenvector) must be found. This is accomplished by projecting the desired eigenvector into the range space of achievable eigenvectors.

Using only the first n rows (n equals the dimension of the system's eigenvectors) of the (n+m) by (n+m-r) null space matrix for expression (2.4), we first must find the coefficient vector $\bar{q}$ which, when premultiplied by the first n rows of the null space matrix, yields the optimal eigenvector. Note that the first n rows of the null space matrix for expression (2.4) defines the range space of the achievable eigenvectors as found in the upper partition of expression (2.3). Denoting the desired eigenvector and the optimal eigenvector associated with the eigenvalue λ_i of expression (2.4) as $\bar{p}_{di}$ and $\bar{p}_{oi}$, respectively, and denoting the first n rows of the null space of expression (2.4) as $[p_i]$, our task is to find the coefficient vector $\bar{q}$ such that

$$\bar{p}_{di} \approx \bar{p}_{oi} = [p_i]\bar{q}$$

The symbol ≈ indicates the quantities are nearly equal, or as close to being equal as possible. If we can find the matrix which, when multiplied by $[p_i]$, gives us the identity matrix, or a form of it, we can solve for $\bar{q}$. Calling such a matrix the pseudoinverse of $[p_i]$, and using the notation $[p_i]^+$ to represent this pseudoinverse, we get

$$[p_i]^+\bar{p}_{di} \approx [p_i]^+\bar{p}_{oi} = [p_i]^+[p_i]\bar{q} = \bar{q}$$

Ideally, $\bar{p}_{oi}$ equals $\bar{p}_{di}$. With this in mind, we will use $\bar{q} = [p_i]^+\bar{p}_{di}$ as the defining relation for $\bar{q}$. This leaves $\bar{p}_{oi} = [p_i][p_i]^+\bar{p}_{di}$ (Strang, 1976: 132). Because $[p_i]$ is dimensioned n by (n+m−r), $[p_i]^+$ is dimensioned (n+m−r) by n. Also, because $\bar{p}_{di}$ and $\bar{p}_{oi}$ both are dimensioned n by 1, this leaves $\bar{q}$ to be dimensioned (n+m−r) by 1. We can see that $\bar{q}$ is that vector which projects the desired eigenvector onto the column space of the first n rows of the null space of expression (2.4); $\bar{q}$ finds the optimal eigenvector which most clearly matches the desired eigenvector. Because $\bar{q}$ projects $\bar{p}_{di}$ into the range space defined by $[p_i]$, it can also be used to find the entire vector

$$\begin{bmatrix} \bar{p}_{oi} \\ \hline G\bar{p}_{oi} \end{bmatrix}$$

This is the optimal vector in the form of expression (2.3) which most nearly matches $\bar{p}_i$ to the desired eigenvector $\bar{p}_{di}$. The next section will show how to find the pseudoinverse of a matrix.

Singular Value Decomposition.

An important tool of numerical linear algebra is the singular value decomposition (SVD) (Klema and Laub, 1980: 166). As alluded to earlier, with the SVD, one can ascertain several important characteristics of linear systems. Important for our applications are the ability to find the null space and pseudoinverse of a matrix.

For any n by m matrix A with rank r, the SVD theorem states that A can be decomposed such that

$$A = U\Sigma V^T \qquad (2.6)$$

where U is an n by n orthogonal matrix, V is an m by m orthogonal matrix, and

$$\Sigma = \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix}$$

Here, $S = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$ with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0$.

The proof of the SVD theorem (Klema and Laub, 1980: 166-167) is as follows. Because the matrix product of A^T and A gives a positive semi-definite matrix (Strang, 1976: 244), that is $A^TA \geq 0$, then all of the eigenvalues of A^TA are greater than or equal to zero (Strang, 1976: 244). In other words, the spectrum of A^TA (dimensioned m by m) is within the range of zero to $+\infty$. We will denote this positive spectrum by $(\sigma_i^2,\ i = 1, 2, \ldots, m)$ and call this set the singular values of A^TA. Here, each σ_i is the positive square root of the eigenvalue λ_i of A^TA. Ordering the singular values such that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0 = \sigma_{r+1} = \ldots = \sigma_m$, we can introduce a corresponding set of orthogonal eigen-

vectors $\bar{v}_1, \ldots, \bar{v}_m$. We know that A^TA has orthogonal eigenvectors because A^TA is symmetric, and every symmetric matrix has an orthogonal set of eigenvectors (Strang, 1976: 135-136). If we group these orthogonal eigenvectors into two matrices

$$V_1 = [\bar{v}_1, \ldots, \bar{v}_r]$$
$$V_2 = [\bar{v}_{r+1}, \ldots, \bar{v}_m]$$

and define $S = \text{diag}(\sigma_1, \ldots, \sigma_r)$, then we can extend the defining relation for an eigenvector so that we can write

$$A^TAV_1 = V_1S^2 \tag{2.7}$$

Recalling that the product of an orthogonal matrix and its transpose yields the identity matrix (Strang, 1976: 120), and because V_1 is orthogonal, we can write $V_1^TA^TAV_1 = S^2$. Premultiplying and postmultiplying both sides of this expression by S^{-1} yields

$$S^{-1}V_1^TA^TAV_1S^{-1} = I \tag{2.8}$$

Because the columns of the orthogonal matrix V_2 are the eigenvectors corresponding to the zero-valued eigenvalues of A^TA, we can again extend the defining relation for an eigenvector as we did in Eq (2.7) and get

$$A^TAV_2 = V_2(0)$$

Again, recognizing the properties of orthogonal matrices, we get

$$V_2^T A^T A V_2 = 0$$

which gives

$$AV_2 = 0 \qquad (2.9)$$

Here we can see that V_2 defines the null space of A. Now introduce the matrix U_1 such that

$$U_1 = AV_1 S^{-1} \qquad (2.10)$$

From Eq (2.8) we have $U_1^T U_1 = I$. If we choose any matrix U_2 such that $U = (U_1, U_2)$ is orthogonal, and because the matrices V_1 and V_2 can be considered to define the orthogonal matrix $V = (V_1, V_2)$, then we can find the matrix product $U^T AV$. Since

$$U^T = \begin{bmatrix} U_1^T \\ \\ U_2^T \end{bmatrix}$$

we have

$$U^T AV = \begin{bmatrix} U_1^T \\ \\ U_2^T \end{bmatrix} A \, (V_1 \, , \, V_2)$$

$$= \begin{bmatrix} U_1^T A \\ \\ U_2^T A \end{bmatrix} (V_1 \, , \, V_2)$$

$$U^TAV = \begin{bmatrix} U_1^TAV_1 & U_1^TAV_2 \\ U_2^TAV_1 & U_2^TAV_2 \end{bmatrix} \qquad (2.11)$$

From Eq (2.10) we have $AV_1 = U_1S$. This gives

$$U_1^TAV_1 = U_1^TU_1S = S \qquad (2.12)$$

From Eq (2.9), we get

$$U_1^TAV_2 = 0 \qquad (2.13)$$

and

$$U_2^TAV_2 = 0 \qquad (2.14)$$

Also, $AV_1 = U_1S$, so that

$$U_2^TAV_1 = U_2^TU_1S = 0 \qquad (2.15)$$

Combining Eqs (2.11) through (2.15) yields

$$U^TAV = \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} = \Sigma \qquad (2.16)$$

Premultiplying both sides of Eq (2.16) by U, and post-multiplying both sides by V^T gives

$$A = U\Sigma V^T \qquad (2.6)$$

as desired.

Again, the numbers $\sigma_1,\ldots,\sigma_r$ along with $\sigma_{r+1} = 0,\ldots,\sigma_n = 0$ are called the singular values of A. The singular values are the positive square roots of the eigenvalues of

the matrix product A^TA. The columns of U are known as the left singular vectors of A, and are the orthonormal eigenvectors of AA^T. Similarly, the columns of V are called the right singular vectors of A, and are the orthonormal eigenvectors of A^TA (Klema and Laub, 1980: 167).

To see in more detail how SVD can be used to find the null space of a matrix (Reid, 1983: 452-453), partition Eq (2.6) as follows.

$$A = [U_1 \ , \ U_2] \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \qquad (2.17)$$

With the singular values in the submatrix S in decreasing order of magnitude, and with the number of non-zero elements of S equal to the rank of A, this partition is such that U_1 has r orthogonal columns $\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_r$, and U_2 has (n-r) orthogonal columns $\bar{u}_{r+1}, \ldots, \bar{u}_n$. Also, the orthogonal matrix V has been partitioned such that V_1 has r orthogonal columns $\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_r$, and V_2 has (m-r) orthogonal columns $\bar{v}_{r+1}, \ldots, \bar{v}_m$.

Because the product of any orthogonal matrix with its transpose yields the identity matrix, we can postmultiply both sides of Eq (2.17) by V and get

$$AV = [U_1 \ , \ U_2] \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} = [U_1 S \ , \ 0]$$

or,

$$A[V_1 \ , \ V_2] = [AV_1 \ , \ AV_2] = [U_1 S, 0]$$

Because the n by r submatrix AV_1 equals the n by r submatrix

$U_1 S$, this leaves the n by (m-r) submatrix AV_2 equal to the zero matrix. Therefore the (m-r) orthogonal columns of V_2 form an orthogonal basis for the m by (m-r) dimensioned null space of A.

As was shown in the second section of this chapter, the projection $\bar{p}$ of a vector $\bar{b}$ onto the column space of a matrix A is $\bar{p} = AA^+\bar{b}$ (Strang, 1976: 132). Here, A^+ is the pseudoinverse of A. With $A = U\Sigma V^T$, we find the pseudoinverse by finding that matrix which, when multiplied by A, gives the identity matrix in the first r rows and columns, and zeroes in the remaining rows and columns of the resulting matrix. Here, r equals the rank of the A matrix. By using the characteristics of orthogonal matrices, it is simple to see that if we define

$$\Sigma^+ = \begin{bmatrix} \text{diag } (1/\sigma_1, \ldots, 1/\sigma_r) & 0 \\ 0 & 0 \end{bmatrix}$$

then

$$V\Sigma^+U^TU\Sigma V^T = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \qquad (2.18)$$

where I is dimensioned r by r, and the entire matrix is m by m. Since $A = U\Sigma V^T$, the left side of Eq (2.18) can be written as $V\Sigma^+U^TA$. From this, it is apparent that $V\Sigma^+U^T$ is that matrix we are looking for. That is, the pseudoinverse is given as

$$A^+ = V\Sigma^+U^T$$

Therefore, the projection $\bar{p}$ of a vector $\bar{b}$ onto the column space of the matrix A is

$$\bar{p} = AA^{+}\bar{b}$$

or

$$\bar{p} = AV\Sigma^{+}U^{T}\bar{b}$$

III. Verifying MODES - Two Check Cases

AFTI/F-16.

Sobel and Shapiro present an application of the eigenstructure assignment method to decouple the pitch attitude and flight path angle for the AFTI/F-16 aircraft (Sobel and Shapiro, 1985: 181-187).

Examining the equation $\dot{\bar{x}} = A\bar{x} + B\bar{u}$, with the state vector $\bar{x}$ and control $\bar{u}$ defined as

$$\bar{x} = \begin{bmatrix} \gamma \\ q \\ \Theta \end{bmatrix} \qquad \bar{u} = \begin{bmatrix} \delta_e \\ \delta_f \end{bmatrix}$$

the A and B matrices are

$$A = \begin{bmatrix} 0 & 0.00665 & 1.3411 \\ 0 & -0.86939 & 43.223 \\ 0 & 0.99335 & -1.3411 \end{bmatrix} \quad B = \begin{bmatrix} 0.16897 & 0.25183 \\ -17.251 & -1.5766 \\ -0.16897 & -0.25183 \end{bmatrix}$$

Here, the control inputs δ_e and δ_f are the elevator and flap deflections, respectively; γ is the flight path; q is the pitch rate; and Θ is the pitch angle. In agreement with Sobel and Shapiro, MODES gives the eigenvalues of this system to be 0.0, 5.452, and -7.662. Sobel and Shapiro's objective is to decouple the flight path and pitch modes to produce closed-loop eigenvalues

$$\lambda_{sp} = -5.6 \pm 4.2i \quad \lambda_3 = -1.0$$

where the desired closed-loop eigenvectors for the short period mode are given by

$$\begin{bmatrix} 0 \pm 0i \\ 1 \pm Xi \\ X \pm 1i \end{bmatrix}$$

and the desired closed-loop flight path mode eigenvector is given as

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

The X's in the short period eigenvectors represent arbitrary values. Because MODES is designed for the user to completely specify the eigenstructure, we will use the actual values found by Shapiro and Sobel for these two eigenvector components. Now, with the short period mode eigenvector given as

$$\begin{bmatrix} 0 + 0i \\ 1 \mp 9.5i \\ -0.9286 \pm 1i \end{bmatrix}$$

we can verify that MODES finds the correct feedback gain matrix G and the correct matrix $\tilde{A}$, where

$$\tilde{A} = A + BG \qquad (2.1)$$

When this problem was solved using the IMSL computer library, the feedback gain matrix G and the closed-loop system matrix $\tilde{A}$ satisfying the equation

$$\dot{\bar{x}} = \tilde{A}\bar{x} \qquad (3.1)$$

are given as

$$G = \begin{bmatrix} 2.2147 & 0.6405 & 6.2141 \\ -5.4568 & -0.4561 & -9.4948 \end{bmatrix}$$

and

$$\tilde{A} = \begin{bmatrix} -1.0 & 0.0 & 0.0 \\ -29.6 & -11.2 & -49.0 \\ 1.0 & 1.0 & 0.0 \end{bmatrix}$$

Rounding off to the same number of significant figures, MODES calculates these matrices to be

$$G = \begin{bmatrix} 2.2145 & 0.6405 & 6.2136 \\ -5.4567 & -0.4562 & -9.4943 \end{bmatrix}$$

and

$$\tilde{A} = \begin{bmatrix} -1.0 & 0.0 & 0.0 \\ -29.6 & -11.2 & -49.0 \\ 1.0 & 1.0 & 0.0 \end{bmatrix}$$

This is in excellent agreement, showing MODES does in

fact generate the proper gains.

F-15.

The second case used to verify the program MODES involves the flight control system for a flight-test maneuver autopilot (Alag and Duke, 1986: 441-445). Considering only the lateral modes, we have the A matrix given as

$$A = \begin{bmatrix} -0.2337 & 0.0358 & -0.9994 & 0.0387 \\ -40.0103 & -2.1420 & 1.2406 & 0.0 \\ 9.0098 & -0.0340 & -0.6040 & 0.0 \\ 0.0 & 1.0 & 0.0358 & 0.0 \end{bmatrix}$$

The B matrix is

$$B = \begin{bmatrix} -0.0022 & -0.0388 \\ 13.5934 & -1.4674 \\ 0.1488 & -4.5577 \\ 0.0 & 0.0 \end{bmatrix}$$

For these matrices, the state vector $\bar{x}$ and control vector $\bar{u}$ are

$$\bar{x} = \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} \qquad \bar{u} = \begin{bmatrix} \delta_a \\ \delta_r \end{bmatrix}$$

Here, β is the yaw angle, p is the roll rate, r is the yaw rate, ϕ is the roll angle, and δ_a and δ_r are the control

inputs for aileron and rudder deflections. Alag and Duke show the open-loop eigenvalues of this system to be

Dutch Roll:	-0.4088 ± 3.2303i
Roll Subsidence:	-2.1413
Spiral:	-0.0209

Note that MODES shows that Alag and Duke have mislabeled the Dutch roll eigenvalues as the short-period eigenvalues. The desired closed-loop eigenvalues are

Dutch Roll:	-2.0 ± 4.0i
Roll Subsidence:	-4.0
Spiral:	-0.002

The desired closed-loop eigenvectors are

$$\text{Dutch Roll:} \begin{bmatrix} 1 \pm Xi \\ 0 \pm 0i \\ X \pm 1i \\ 0 \pm 0i \end{bmatrix}$$

$$\text{Roll Subsidence:} \begin{bmatrix} 0 \\ 1 \\ 0 \\ X \end{bmatrix} \qquad \text{Spiral:} \begin{bmatrix} 0 \\ X \\ X \\ 1 \end{bmatrix}$$

where the variable X represents arbitrary values.

Alag and Duke state that their optimal closed-loop eigenvectors are

$$\text{Dutch Roll:} \begin{bmatrix} .99 \pm 2.71i \\ -0.21 \mp 0.00i \\ 12.35 \pm 1.09i \\ -0.35 \pm 0.09i \end{bmatrix}$$

$$\text{Roll Subsidence:} \begin{bmatrix} -0.0068 \\ 1.0 \\ 0.0019 \\ -0.25 \end{bmatrix} \qquad \text{Spiral:} \begin{bmatrix} 0.0 \\ -0.0034 \\ 0.0388 \\ 1.0 \end{bmatrix}$$

As with the AFTI/F-16 case, we will select as our arbitrary X's those values found by Alag and Duke. This leaves our desired closed-loop eigenvectors

$$\begin{matrix}\text{Dutch} \\ \text{Roll:}\end{matrix}\begin{bmatrix} 1 \pm 2.71i \\ 0 \pm 0i \\ 12.35 \pm 1i \\ 0 \pm 0i \end{bmatrix}$$

$$\begin{matrix}\text{Roll} \\ \text{Subsidence:}\end{matrix}\begin{bmatrix} 0 \\ 1 \\ 0 \\ -0.25 \end{bmatrix} \qquad \text{Spiral:}\begin{bmatrix} 0 \\ -0.0034 \\ 0.0388 \\ 1 \end{bmatrix}$$

With this as our specified eigenstructure, we can see if MODES does calculate the proper feedback gain matrix. Alag and Duke show that the feedback gain matrix should be

$$G = \begin{bmatrix} 2.7167 & -0.1422 & -0.0147 & -0.0046 \\ -2.0771 & -0.0483 & 0.7094 & -0.0329 \end{bmatrix}$$

MODES calculates the feedback gain matrix to be

$$G = \begin{bmatrix} 2.7168 & -0.1422 & -0.0147 & -0.0046 \\ -2.0769 & -0.0483 & 0.7094 & -0.0329 \end{bmatrix}$$

This leaves $\tilde{A} = A + BG$ to be

$$\tilde{A} = \begin{bmatrix} -0.1591 & 0.0380 & -1.0269 & 0.0400 \\ -0.0326 & -4.0036 & -0.0003 & -0.0136 \\ 18.8800 & 0.1651 & -3.8393 & 0.1494 \\ 0.0 & 1.0 & 0.0358 & 0.0 \end{bmatrix}$$

MODES shows the optimal closed-loop eigenvectors to be

$$\text{Dutch Roll:} \begin{bmatrix} .99 \pm 2.71i \\ -0.021 \mp 0.00i \\ 12.33 \pm 1.00i \\ -0.035 \mp 0.09i \end{bmatrix}$$

$$\text{Roll Subsidence:} \begin{bmatrix} -0.0068 \\ 1.0 \\ 0.0019 \\ -0.25 \end{bmatrix} \qquad \text{Spiral:} \begin{bmatrix} 0.0 \\ -0.0034 \\ 0.0388 \\ 1.0 \end{bmatrix}$$

Note the differences in the Dutch Roll eigenvectors found by MODES and those presented by Alag and Duke. The IMSL library routine EIGRF agrees with MODES. Again, as with the AFTI/F-16 case, we have excellent agreement between MODES and a known problem.

IV. Discussion of Routines

SVD.

The subroutine SVD is a modified version of the routine CSVD, found as "Algorithm 358-Singular Value Decomposition of a Complex Matrix" (Businger and Golub, 1980: 358-P-1, 358-P-2) in Collected Algorithms from CACM. CSVD is written for complex matrices. SVD is a modification of CSVD to address real matrices.

As found in Chapter II, the singular value decomposition of the n by m matrix A is

$$A = U\Sigma V^T \qquad (2.6)$$

Here, the matrix U consists of the n orthonormalized eigenvectors asssociated with the n eigenvalues of AA^T. The matrix V consists of the m orthonormalized eigenvectors of A^TA. The diagonal elements of Σ are the non-negative square roots of the eigenvalues of A^TA, and are called the singular values.

Although one might decide to calculate the singular values of A by finding the eigenvalues of A^TA, needless errors can result. As an example, let

$$A = \begin{bmatrix} 1 & 1 \\ y & 0 \\ 0 & y \end{bmatrix}$$

Here,

$$A^TA = \begin{bmatrix} 1+y^2 & 1 \\ 1 & 1+y^2 \end{bmatrix}$$

and the singular values are $\left(2+y^2\right)^{1/2}$ and $|y|$, showing that A has rank 2. However, if $|y|$ is less than the floating point accurracy of the computer used then

$$A^TA = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

and the singular values would be given as $\sqrt{2}$ and 0, indicating that A has rank 1. Therefore, by working with A^TA directly, we are apt to introduce unnecessary errors (Klema and Laub, 1980: 167). SVD uses Householder transformations to reduce A to bidiagonal form, and then the QR algorithm to find the singular values of this bidiagonal matrix. For this routine, we assume that A is dimensioned m by n, and that $m \geq n$. If $m < n$ we perform SVD on the transpose of the matrix.

Reduction to Bidiagonal Form (Golub and Reinsch, 1970: 404-405). A Householder transformation is an orthogonal and symmetric matrix D of the form

$$D = I - \frac{2}{||\bar{v}||^2} \bar{v}\,\bar{v}^T$$

where $\bar{v}$ is an orthogonal transformation vector. In our application, we will construct two finite sequences of Householder transformations

$$P^{(k)} = I - 2\bar{x}^{(k)}\bar{x}^{(k)^T} \qquad (k=1,2,\ldots,n)$$

and

$$Q^{(k)} = I - 2\bar{y}^{(k)}\bar{y}^{(k)^T} \qquad (k=1,2,\ldots,n-2)$$

where $\bar{x}^{(k)}$ and $\bar{y}^{(k)}$ have been normalized, so that

$$\bar{x}^{(k)^T}\bar{x}^{(k)} = \bar{y}^{(k)^T}\bar{y}^{(k)} = 1$$

This will give us

$$P^{(n)}\ldots P^{(1)}AQ^{(1)}\ldots Q^{(n-2)} = \begin{bmatrix} q_1 & e_2 & 0 & \cdots & 0 \\ & q_2 & e_3 & & \vdots \\ & & \ddots & \ddots & 0 \\ & & & & e_n \\ 0 & & & & q_n \\ \hline & & 0 & & \end{bmatrix}$$

$$= J^{(o)}$$

Here, $J^{(o)}$ is an upper bidiagonal matrix dimensioned m by n. The upper partition shown is dimensioned n by n, and the lower partition is (m-n) by n.

Letting $A^{(1)} = A$, define

$$A^{(k+1/2)} = P^{(k)}\,A^{(k)} \qquad (k=1,2,\ldots,n)$$

$$A^{(k+1)} = A^{(k+1/2)}Q^{(k)} \qquad (k=1,2,\ldots,n)$$

This leaves

$$A^{(k+1)} = P^{(k)}A^{(k)}Q^{(k)} \qquad (k=1,2,\ldots,n-2)$$

Using the notation $a_{ij}^{(k)}$ to represent the element in the i^{th} row and j^{th} column of the matrix $A^{(k)}$, we will determine $P^{(k)}$ such that

$$a_{ik}^{(k+1/2)} = 0 \qquad (i=k+1,\ldots,m)$$

and $Q^{(k)}$ such that

$$a_{kj}^{(k+1)} = 0 \qquad (j=k+2,\ldots,n)$$

This leaves $J^{(o)}$ to be bidiagonal, as desired, with zeroes everywhere except along the main diagonal, and along the first diagonal above the main diagonal. If the matrix $J^{(o)}$ has the singular value decomposition

$$J^{(o)} = G\Sigma H^T$$

then

$$P^{(n)}\ldots P^{(1)}AQ^{(1)}\ldots Q^{(n-2)} = G\Sigma H^T$$

Because the $P^{(k)}$ and $Q^{(k)}$ matrices are orthogonal, we can write

$$A = P^{(1)^T}\, P^{(2)^T}\ldots\, P^{(n)^T}G\Sigma H^T Q^{(n-2)^T}\ldots Q^{(1)^T}$$

Recalling that the Householder transformation produces matrices which are both orthogonal and symmetric, we have

$$A = P^{(1)} \ldots P^{(n)} G \Sigma H^T Q^{(n-2)^T} \ldots Q^{(1)^T}$$

For notational convenience, we will denote the products $P^{(1)} \ldots P^{(n)}$ and $Q^{(1)} \ldots Q^{(n-2)}$ as P and Q, respectively. This leaves

$$A = PG\Sigma H^T Q^T$$

Strang shows that multiplication by an orthogonal matrix T preserves lengths (Strang, 1976: 121). That is,

$$||T\bar{x}|| = ||\bar{x}||$$

for every vector $\bar{x}$. This multiplication also preserves inner products:

$$(T\bar{x})^T(T\bar{y}) = \bar{x}^T\bar{y}$$

for all vectors $\bar{x}$ and $\bar{y}$. With this, and because P and Q^T are orthogonal, we find that the singular values of $J^{(o)}$ are the same as those of A. With

$$U\Sigma V^T = A = PG\Sigma H^T Q^T = PJ^{(o)}Q^T$$

we have $U = PG$ and $V = QH$. Therefore, given the singular value decomposition of $J^{(o)}$, we can find the singular value decomposition of A. Our next task is to perform the singular value decomposition of the bidiagonal matrix $J^{(o)}$.

Singular Value Decomposition of the Bidiagonal Matrix (Golub and Reinsch, 1970:405-406). Using a variation of the

QR Algorithm, we diagonalize the matrix $J^{(o)}$ iteratively such that $J^{(o)}$ becomes $J^{(1)}$, which becomes $J^{(2)}$,..., which becomes Σ. Here,

$$J^{(i+1)} = S^{(i)T} J^{(i)} T^{(i)}$$

where $S^{(i)}$ and $T^{(i)}$ are orthogonal. The matrices $S^{(i)}$ are chosen so that all $J^{(i)}$ are bidiagonal, while the matrices $T^{(i)}$ are chosen so that the sequence $M^{(i)} = J^{(i)T} J^{(i)}$ converges to a diagonal matrix. For notational convenience, let

$$J = J^{(i)}, \quad \tilde{J} = J^{(i+1)}, \quad S = S^{(i)}, \quad T = T^{(i)},$$
$$M = J^T J, \quad \tilde{M} = \tilde{J}^T \tilde{J}$$

Now we will use a method called Givens rotations to eliminate off-diagonal terms so as to transform J to a diagonal matrix. We achieve this transformation from J to $\tilde{J}$ by applying a Givens rotation to J first from the right, then from the left, and continuing to alternate such that

$$\begin{aligned} \tilde{J} &= S_n^T S_{(n-1)}^T \cdots S_2^T J T_2 T_3 \cdots T_n \\ &= S^T J T \end{aligned} \tag{4.1}$$

where,

$$S_k = \begin{bmatrix} 1 & 0 & & & & & \\ 0 & \ddots & \ddots & & & & \\ & & 1 & & & & \\ & & & \cos(\theta_k) & -\sin(\theta_k) & & \\ & & & \sin(\theta_k) & \cos(\theta_k) & & \\ & & & & & 1 \quad \ddots \quad 0 & \\ & & & & & 0 \quad \ddots \quad 1 & \end{bmatrix}$$

and T_k is similarly defined with ϕ_k instead of θ_k. The $\cos(\theta_k)$ terms are on the main diagonal, in the $S_k(k-1,k-1)$ and $S_k(k,k)$ positions.

Because we apply the Givens rotation first from the right, we first deal with the T_2 matrix. Denoting the non-zero, non-unity elements as #, we have, for a sample 5 by 5 matrix,

$$JT_2 = \begin{bmatrix} \# & \# & 0 & 0 & 0 \\ 0 & \# & \# & 0 & 0 \\ 0 & 0 & \# & \# & 0 \\ 0 & 0 & 0 & \# & \# \\ 0 & 0 & 0 & 0 & \# \end{bmatrix} \begin{bmatrix} \# & \# & 0 & 0 & 0 \\ \# & \# & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \# & \# & 0 & 0 & 0 \\ \# & \# & \# & 0 & 0 \\ 0 & 0 & \# & \# & 0 \\ 0 & 0 & 0 & \# & \# \\ 0 & 0 & 0 & 0 & \# \end{bmatrix}$$

We see that where J was bidiagonal, JT_2 is almost bidiagonal, with an additional non-zero element in the 2,1 position. The angle ϕ_2 associated with T_2 is, at this

point, arbitrary. All other angles are chosen so that $\tilde{J}$ has the same form as J. The angle θ_2 associated with S_2 is chosen so that the product $S_2^T J T_2$ annihilates the element in the 2,1 position.

Here,

$$S_2^T J T_2 = \begin{bmatrix} \# & \# & 0 & 0 & 0 \\ \# & \# & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \# & \# & 0 & 0 & 0 \\ \# & \# & \# & 0 & 0 \\ 0 & 0 & \# & \# & 0 \\ 0 & 0 & 0 & \# & \# \\ 0 & 0 & 0 & 0 & \# \end{bmatrix}$$

$$= \begin{bmatrix} \# & \# & \# & 0 & 0 \\ X & \# & \# & 0 & 0 \\ 0 & 0 & \# & \# & 0 \\ 0 & 0 & 0 & \# & \# \\ 0 & 0 & 0 & 0 & \# \end{bmatrix} = \begin{bmatrix} \# & \# & \# & 0 & 0 \\ 0 & \# & \# & 0 & 0 \\ 0 & 0 & \# & \# & 0 \\ 0 & 0 & 0 & \# & \# \\ 0 & 0 & 0 & 0 & \# \end{bmatrix}$$

Again, the angle θ_2 is chosen such that the X in the 2,1 position equals zero. Now we have introduced a non-zero element in the 1,3 position. To make this element zero, the angle ϕ_3 associated with T_3 is appropriately selected. Similarly to before, the product $S_2^T J T_2 T_3$ will introduce a non-zero entry in the 3,2 position. Continuing to alternate these multiplications, we finally find that S_n^T annihilates a non-zero entry in the n,(n-1) position, while generating no new non-zero entries. Therefore, $\tilde{J}$ is also bidiagonal.

Since $\tilde{J} = S^T J T$

$$\tilde{M} = \tilde{J}^T\tilde{J} = T^TJ^TSS^TJT$$

Because S is orthogonal, $\tilde{M} = T^TJ^TJT = T^TMT$, and $\tilde{M}$ is a tri-diagonal matrix, as is M. We now will see that the selection of the angle ϕ_2 associated with the matrix T_2, up to now an arbitrary angle, can be chosen so that the transition from M to $\tilde{M}$ is a QR transformation with a given shift s. A QR transformation is an algorithm used to calculate the eigenvalues and eigenvectors of a matrix. Convergence of this algorithm can be accelerated if we properly shift the eigenvalues of the matrix M by an amount s (Press, et al, 1986: 356-363).

The usual QR algorithm with shift s is described as:

$$\begin{aligned} M - sI &= T_sR_s \\ R_sT_s + sI &= \tilde{M}_s \end{aligned} \qquad (4.2)$$

where $T_s^TT_s = I$, and R_s is an upper triangular matrix.

With $T_s^TT_s = I$, we have

$$R_sT_s = T_s^TT_sR_sT_s = T_s^T(T_sR_s)T_s = T_s^T(M-sI)T_s$$

Therefore,

$$R_sT_s + sI = T_s^T(M-sI)T_s + sI = \tilde{M}_s$$

or,

$$T_s^TMT_s - T_s^T(sI)T_s + sI = \tilde{M}_s$$

This gives

$$T_s^TMT_s = \tilde{M}_s$$

Golub and Reinsch (Golub and Reinsch, 1970:406) show that we do not have to compute Eq (4.2) explicitly, but that we can implicitly perform the shift. For now, let T be an arbitrary matrix such that the elements of the first column of T are equal to the elements of the first column of T_s. That is,

$$T(k,1) = T_s(k,1) \qquad (k=1,2,\ldots,n)$$

Also, we must have $T^TT = I$. Given this, Golub and Reinsch present the following theorem, offered without proof:

If i) $\tilde{M} = T^TMT$,
ii) $\tilde{M}$ is tri-diagonal,
and iii) the sub-diagonal elements of M are zero,

then $\tilde{M} = D\tilde{M}_sD$ where D is a diagonal matrix whose diagonal elements are ±1.

By choosing the angle θ_2 associated with the matrix T_2 such that the first column of T_2 is proportional to the first column of M-sI, then the first column of the product $T = T_2T_3\ldots T_n$ is also proportional to the first column of M-sI. Remembering that the first column of T is identical to the first column of T_s, then, if the sub-diagonal of M does not contain any non-zero entry, the theorem conditions are met. Therefore, T is identical to T_s (up to a scaling of column ±1). The transition in Eq (4.1) is equivalent to the QR transformation of J^TJ with a given shift s.

The shift parameter s is given as an eigenvalue of the lower 2 by 2 minor of M. For this choice of s, the method converges globally, and usually converges cubically. Now we must determine when we have found the J which ends our

search.

Test for Convergence (Golub and Reinsch, 1970:407). With the matrix J and a prescribed tolerance δ, we are ready to examine the system for convergence. If $|e_n| \leq \delta$ then we accept $|q_n|$ as a singular value, and we reduce the order of the matrix by one. If $|e_k| \leq \delta$ for $k \neq n$, then we break the matrix into two separate blocks, and independently compute the singular values of both blocks.

If $q_k = 0$, then we have at least one singular value equal to zero. Provided we have no roundoff error, the matrix will break if we perform a shift of zero. Therefore, assume that at some stage, $|q_k| \leq \delta$. At this stage, we apply an extra sequence of Givens rotations to J from the left involving rows k and k+1, rows k and k+2,..., and rows k and n. Doing this, we have formed the matrix

$$\mathcal{J} = \begin{bmatrix} q_1 & e_2 & & & & & 0 \\ & \ddots & \ddots & & & & \\ & & & e_k & & & \\ & & & q_k & 0 & & \\ & & & \delta_{k+1} & q_{k+1} & e_{k+2} & \\ & & & \vdots & \ddots & \ddots & \\ & & & \vdots & & \ddots & e_n \\ 0 & & & \delta_n & & & q_n \end{bmatrix}$$

With the construction of the $\mathcal{J}$ matrix with the extra sequence of Givens rotations, orthogonality provides that

$$q_k^2 + \delta_{k+1}^2 + \ldots + \delta_n^2 = q_k^2 \leq \delta^2$$

Our remaining task is to choose an appropriate δ such that

all of the δ_k terms are negligible. To do this, we will let ε_o be the machine precision of the computer used, and we will use the uniform norm as a relative "size" of a matrix where the uniform norm is defined as (Ralston and Rabinowitz, 1978: 8):

$$\text{Uniform norm of } A = ||A||_\infty = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$$

By choosing $\delta = \varepsilon_o ||J^{(o)}||_\infty$, we are ensured that all $|\delta_k|$ are less than $\varepsilon_o ||J^{(o)}||_\infty$. The elements of $\mathcal{J}$ which are less than or equal to this are neglected. Therefore, $\mathcal{J}$ breaks into two parts, and these parts are treated independently.

Setting Up Matrices for Singular Value Decomposition.

As was discussed in Chapter II, we need to perform the singular value decomposition on a matrix for two reasons in MODES: 1) to find the null space of the augmented matrix $[A-\lambda_i I, B]$ so as to find the range space of achievable eigenvectors of the system, and 2) to find the pseudoinverse of the range space of the achievable eigenvectors so that we can project the desired eigenvectors into that range space.

Null Space of the Augmented Matrix. For real eigenvalues, the formation of the basic augmented matrix $[A-\lambda_i I, B]$ is straight forward. Where the elements of a sample 3 by 3 matrix A and a sample 3 by 2 matrix B are written as a_{ij} and b_{ij}, respectively, the basic augmented matrix is simply

$$\begin{bmatrix} a_{11}-\lambda_i & a_{12} & a_{13} & b_{11} & b_{12} \\ a_{21} & a_{22}-\lambda_i & a_{23} & b_{21} & b_{22} \\ a_{31} & a_{32} & a_{33}-\lambda_i & b_{31} & b_{32} \end{bmatrix}$$

Note that for an n by n A matrix, and an n by m B matrix, the basic augmented matrix is dimensioned n by (n+m).

Because the routine SVD is written for real matrices only, the augmented matrix involving complex eigenvalues must be altered. If we write the complex eigenvalue

$$\lambda_k = \lambda_{rk} + j\lambda_{ik}$$

where λ_{rk} is the real component of the k^{th} eigenvalue and λ_{ik} is the complex component, the altered augmented matrix can be written as (Silverthorn and Reid, 1980: 1207)

$$\left[\begin{array}{c|c|c|c} A - \lambda_{rk}I & \lambda_{ik}I & B & 0 \\ \hline -\lambda_{ik}I & A-\lambda_{rk}I & 0 & B \end{array}\right]$$

For an n by n matrix A and an n by m matrix B, the altered augmented matrix is dimensioned 2n by 2(n+m).

In chapter II, when calculating the null space of the augmented matrix, we represented the null space as a matrix of vectors

$$\begin{bmatrix} \overline{P} \\ \hline \overline{GP} \end{bmatrix}$$

The upper partition contains the closed-loop eigenvectors of the matrix A, and the lower partition is the product of the feedback gain matrix and the matrix of closed-loop eigenvectors. For a complex eigenvalue, we have one column of the complex null space matrix in the form (Silverthorn and Reid, 1980: 1207)

$$\begin{bmatrix} \bar{p}_{r\,k} \\ \bar{p}_{i\,k} \\ \hline G\bar{p}_{r\,k} \\ G\bar{p}_{i\,k} \end{bmatrix}$$

As before, the subscripts rk refer to the real components of the k^{th} eigenvector and the subscripts ik refer to the imaginary components.

The MODES subroutine SETUPC is used to set up the altered augmented matrix, and is used whether the augmented matrix is real or imaginary. Naturally, when SETUP is used with real eigenvectors, $\lambda_{i\,k}$ is set equal to zero.

Because the subroutine SVD is written to perform the singular value decomposition on a matrix which has either more rows than columns or an equal number of rows and columns, and because the augmented matrix will always have more columns than rows, the subroutine SETUPC defines the augmented matrix to be the transpose of the matrix defined above. This way, SVD can perform properly. The only modification required for the routine SVD is in our call statement. We can see that if the singular value decomposition of the matrix A is

$$A = U_A \Sigma_A V_A^T$$

then the singular value decomposition of the transpose of A is

$$A^T = \left\{U_A \Sigma_A V_A^T\right\}^T = V_A \Sigma_A^T U_A^T$$

But, the singular value decomposition of A^T can also be written as

$$A^T = U_{A^T} \Sigma_{A^T} V_{A^T}^T$$

Therefore, if we want the singular value decomposition of A, but are limited to using A^T, then we can do this provided we recognize that the U_{A^T} and the V_{A^T} matrices associated with the matrix A^T are the same as the V_A and U_A matrices, respectively. With this in mind, when we want the singular value decomposition of the augmented matrix, we switch the locations of U and V in the call statement, and apply SVD to the transpose of the augmented matrix.

Pseudoinverse of the Range Space. After applying SVD to the transpose of the augmented matrix $[A-\lambda_i I, B]$, we use the null space defined by V_2 from Chapter II to find the range space of achievable eigenvectors. When the matrix A is dimensioned n by n, this range space is the first n rows of the null space matrix V_2. If the eigenvalue used in the augmented matrix is complex, the range space is the first 2n rows of V_2. Subroutine V2DEF defines the null space matrix from the singular value decomposition of the transposed augmented matrix, and subroutine V2SDEF defines the range space of achievable eigenvectors from V_2. This range space matrix, called V2S in MODES, is dimensioned n by n when the associ-

ated eigenvalue is real, and it is dimensioned 2n by 2n when the associated eigenvalue is complex.

The subroutine SVD is applied to the square matrix V2S directly. Because SVD is written for matrices which are square, or which have more rows than columns, no special considerations are required here as they were for the augmented matrix.

Calculating the Eigenstructure of a Matrix.

The subroutine EIGV, developed specifically for MODES, is a calling routine which calls the subroutines BALANC, ELMHES, ELTRAN, HQR2, and BALBAK, in that order, to find the eigenvalues and eigenvectors of a general, real matrix. Because these five routines found in the EISPACK manual (Smith, 1976: 200-210, 310-319, 338-348) destroy the original matrix, EIGV also uses the subroutine COPYAB to create a duplicate matrix which is then entered in the series of EISPACK routines. This way we retain the original matrix while computing its eigenstructure. Because our earlier discussion of the subroutine SVD examined in detail a variation of the QR algorithm found in HQR2, we will discuss these routines only in general terms here.

Balancing a Real General Matrix (Parlett and Reinsch, 1969: 293-304). The first of our EISPACK routines, BALANC, is derived from the Algol procedure "balance" by Parlett and Reinsch. Because most eigenvalue programs usually result in errors proportional to the Froebinius norm of the matrix of interest, the purpose of BALANC is to reduce the norm of a matrix without changing its eigenvalues. The Froebinius norm of a matrix A, $||A_F||$, is often used to characterize the "size" of a matrix. The Froebinius norm is defined as:

$$||A_F|| \equiv \left(\sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2 \right)^{1/2}$$

where a_{ij} is the element in the i^{th} row and j^{th} column of A. When two matrices have the same eigenvalues, they are said to be similar (Strang, 1976: 221). Therefore, by performing similarity transformations on the original matrix to find a similar matrix with a reduced norm, we generate a "balanced" matrix whose eigenvalues we are able to calculate with greater accuracy, and whose eigenvalues equal the eigenvalues of the original matrix.

While the eigenvalues of the balanced matrix equal the eigenvalues of the original matrix, the eigenvectors of the two matrices are not equal. Therefore, the subroutine BALBAK is needed to "back transform" the eigenvectors of the balanced matrix into the eigenvectors of the original real, general matrix.

Reducing a General Matrix to an Upper-Hessenberg Matrix (Martin and Wilkinson, 1968: 349-368). Because several algorithms which find the eigenstructure of matrices have fewer calculations to perform when the matrix has an upper-Hessenberg form, we want to transform our balanced matrix to an upper-Hessenberg matrix without destroying the eigenstructure of the original matrix. An upper-Hessenberg matrix is a matrix of the form

$$H = \begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & 0 & X & X & X \\ 0 & 0 & 0 & X & X \end{bmatrix}$$

where the X's represent unspecified values, and where $h_{i,j} = 0 \quad (i > j+1)$. For an n by n general matrix, the number of operations required by the QR method of calculating the eigenstructure (the method employed by HQR2) is on the order of n^3; if the QR method is applied to upper-Hessenberg matrices, the number of calculations required are on the order of n^2 (Strang, 1976: 281). ELMHES uses similarity transformations to convert our balanced matrix into an upper-Hessenberg matrix. As with BALANC, these similarity transformations leave the upper-Hessenberg matrix with eigenvalues equal to the balanced matrix eigenvalues, and therefore equal to the original matrix eigenvalues. However, again as with BALANC, the similarity transformations alter the eigenvectors of the matrix. Therefore, the subroutine ELTRAN is used to accumulate these transformations into a transformation matrix to be used by HQR2 to convert the eigenvectors of the upper-Hessenberg matrix into the eigenvectors of the balanced matrix. From here, as discussed above, BALBAK transforms the eigenvectors of the balanced matrix into the eigenvectors of the original general matrix.

Calculating the Eigenstructure of an Upper-Hessenberg Matrix (Strang, 1976: 280-282). The EISPACK subroutine HQR2 is derived from the Algol procedure hqr2 (Peters and Wilkinson, 1970: 191-197). Using the shifted QR algorithm,

HQR2 finds the eigenvalues and eigenvectors of the upper-Hessenberg matrix in a similar manner to the way SVD finds the eigenstructure of a bidiagonal matrix. Where SVD iterated on the bidiagonal matrix until the off-diagonal elements approach zero, HQR2 uses Householder transformations to iterate on the subdiagonal elements so that they approach zero. This leaves the eigenvalues to be the elements of the main diagonal. The last element of the main diagonal is the one which normally approaches the eigenvalue first. This leaves our transformed matrix A_k to look like

$$A_k = \begin{bmatrix} X & X & X & X & X \\ X & X & X & X & X \\ 0 & X & X & X & X \\ 0 & 0 & X & X & X \\ 0 & 0 & 0 & \varepsilon & \lambda_1 \end{bmatrix}$$

with $\varepsilon << 1$.

At this point, the QR algorithm continues to work on the n-1 by n-1 sub-matrix found in the first n-1 rows and columns of A_k. In this fashion, we continue to find an eigenvalue, reduce the size of the matrix, find another eigenvalue, and repeat this reduce/find eigenvalue process until all eigenvalues have been found. By using back-substitution, we use the zero elements produced by the Householder transformations to find the eigenvectors. That is, given the defining equation of an eigenvalue λ and an eigenvector $\bar{x}$ for a matrix A,

$$A\bar{x} = \lambda\bar{x}$$

we use the information in the last row of the Hessenberg matrix, along with each eigenvalue, to directly find the last component of each eigenvector. Then we use the next-to-last row of the matrix and the last component of the eigenvectors to solve for the next-to-last component of the eigenvectors. We continue to back-substitute in this manner until all of the components of all eigenvectors have been found.

Now we have the eigenstructure of the upper-Hessenberg matrix. By multiplying the matrix of eigenvectors by the accumulated transformations from ELTRAN, found in a single transformation matrix, we get the eigenvectors of the balanced matrix. At this point, our only remaining task is to find the eigenvectors of the original general matrix from the eigenvectors of the balanced matrix. Again, this is performed by the subroutine BALBAK.

Defining and Printing the Eigenstructure.

The subroutine EIGSTR asks the user to define the desired eigenstructure for MODES. When the A matrix of the equation $\dot{\bar{x}} = A\bar{x} + B\bar{u}$ is dimensioned n by n, then the system has n eigenvalues and n eigenvectors. The eigenvectors are stored in a 2n by n array called EVEC, and the eigenvalues are stored in a 2n by 1 array called EVAL.

The first column of the eigenvector array is the eigenvector associated with the first eigenvalue of the eigenvalue array. Similarly, the i^{th} column of the eigenvector array is the eigenvector associated with the i^{th} eigenvalue of the eigenvalue array. The first n elements of any eigenvector column are the real components of the eigenvector, and the last n elements of the eigenvector column are the imaginary components of the eigen-

vector. The j^{th} element ($1 \leq j \leq n$) of the eigenvalue array is the real component of the j^{th} eigenvalue; the $j + n^{th}$ element is the imaginary component of the j^{th} eigenvalue.

By storing the desired eigenstructure in this form, we are prepared to project the desired eigenvectors into the range space of achievable eigenvectors as discussed in Chapter II and in the section "Setting Up Matrices for Singular Value Decompostion" of this chapter.

Because complex eigenvalues and eigenvectors must appear in complex conjugate pairs, EIGSTR automatically defines the complex conjugates.

When MODES calculates the eigenstructure of a matrix, the eigenstructure is stored in a different manner. This different storage is the format used by the EISPACK routines HQR2, ELMHES, ELTRAN, BALANC, and BALBAK (Smith, et al, 1976). Instead of storing the eigenvectors in a 2n by n array as in EIGSTR, these eigenvectors are stored in an n by n array. Also, where EIGSTR stores the eigenvalues in a single 2n by 1 array, these eigenvalues are stored in two separate n by 1 arrays, one array for the real components of the eigenvalues, and the second array for the imaginary components.

When the i^{th} eigenvalue is real, that is when the i^{th} element in the array of imaginary components of the eigenvalues equals zero, then the eigenvector associated with this eigenvalue is found in the i^{th} column of the eigenvector array. When the i^{th} eigenvalue is the first of a pair of complex conjugate eigenvalues, then the i^{th} column of the eigenvector array contains the real components of the associated eigenvector, and the $i+1^{st}$ column contains the imaginary components. Because the $i+1^{st}$ eigenvalue will

be the complex conjugate of the i^{th} eigenvalue, we know that the $i+1^{st}$ eigenvector is the complex conjugate of the i^{th} eigenvector. The subroutine PRINTZ prints the eigenstructure stored in this format.

Calculating the Modified System Matrix.

As was shown in chapter II, the modified closed-loop system matrix $\tilde{A}$ is given as

$$\tilde{A} = A + BG \tag{2.1}$$

Here, the matrices A and B are defined for the linear time invariant state-space model

$$\dot{\bar{x}} = A\bar{x} + B\bar{u} \tag{4.3}$$

The matrix G is the feedback gain matrix required to give the closed-loop system

$$\dot{\bar{x}} = \tilde{A}\bar{x} \tag{3.1}$$

the desired response characteristics as described by the eigenstructure of $\tilde{A}$. The subroutine ATLDEF calculates $\tilde{A}$ from A, B, and G. Chapter II gives the feedback gain matrix G as

$$G = RP_o^{-1} \tag{2.5}$$

Here, the P_o matrix is the matrix of optimal eigenvectors found by projecting the desired eigenvectors into the range space of achievable eigenvectors. As was discussed in Chapter II, the R matrix consists of the lower partition of

the vectors

$$\begin{bmatrix} \bar{p}_{ol} \\ \cdots \\ G\bar{p}_{ol} \end{bmatrix}$$

The upper partition of these vectors are the optimal projection of the desired eigenvectors into the range space of the achievable eigenvectors.

To find the P_o and R matrices, we use subroutine PDDEF to define a desired eigenvector PD from the matrix EVEC which was defined in the subroutine EIGSTR, and the subroutine SVD is used as outlined in chapter II. See "Defining and Printing the Eigenstructure" above for more information on the matrix EVEC.

In the discussion "Setting Up Matrices for Singular Value Decomposition" above, we have stored the range space of achievable eigenvectors in the matrix V2S. As previously stated, we need to project the desired eigenvectors into this range space. Using the methods of chapter II, we do this by performing the singular value decomposition of the matrix V2S, and calculate

$$(PA) = (V2S)V\Sigma^{+}U^{T}(PD)$$

Here, V and U come directly from the subroutine SVD. The subroutine ATRANS defines the transpose of U, U^T. The vector PD is the desired eigenvector. Σ^+ is defined in chapter II to be a diagonal matrix containing the multiplicative inverses of the singular values of V2S in the first r (r=rank) elements of the main diagonal. The subroutine SIGPL defines Σ^+ from the singular value

decomposition of V2S.

With this information, the subroutine AXBEQC performs the required matrix multiplications to find the optimal eigenvectors PA. Subroutine PACHDF defines the matrix of optimal eigenvectors PACH from each optimal eigenvector PA. As shown in chapter II, PA is dimensioned 2(n+m) by 1, and PACH is 2(n+m) by n. As with the matrix EVEC, the first n elements of PA and the first n elements of the columns of PACH are the real components of the optimal eigenvectors, and the last n elements are the imaginary components. The next m elements are the real components of the column vectors of the matrix R. The last m elements are the imaginary components of the column vectors of R.

The subroutine PNRDEF uses the matrix of optimal eigenvectors PACH and rearranges the information in a form suitable for matrix inversion. The matrices formed are the required P_o and R. The n by n matrix P_o is structured as follows. When the i^{th} eigenvector of the modified system is real, the i^{th} column of P_o is that eigenvector. When the i^{th} eigenvector is the first of a pair of complex conjugate eigenvectors, the i^{th} column of P_o contains the real components of that eigenvector while the $i+1^{st}$ column contains the imaginary components. Because any complex eigenvector must have associated with it a complex conjugate, storing one complex eigenvector effectively stores the information of two eigenvectors. The R matrix defined by PNRDEF is structured similarly to P_o.

Using the subroutines INVERT, FACTOR and SUBST, we now find the inverse of P_o, P_o^{-1}. Multiplying R by P_o^{-1}, we have the feedback gain matrix G required to optimize our system's time response characteristics.

Subroutine ATLDEF now multiplies the matrices B and G,

and adds this product to the matrix A to yield the modified system matrix $\tilde{A}$.

Discretizing and Plotting the System Time Response (Reid, 1983: 273-277).

From the linear time-invariant model

$$\dot{\bar{x}}(t) = A\bar{x}(t) + B\bar{u}(t)$$

we have the discrete time state-space model

$$\bar{x}_T(k) = A_T\bar{x}_T(k-1) + B_T\bar{u}_T(k-1) \qquad (4.4)$$

where

$$\bar{u}_T(k) = \bar{u}(t) \qquad (kT \le t \le kT + T)$$

$$A_T \equiv e^{AT} \qquad B_T \equiv \left(\int_0^T e^{A\zeta}d\zeta\right)B$$

T is a given time increment, and k is an integer index indicating the number of time increments which have passed since the starting time t=0. If $\bar{u}_T(k)$ is exact, then

$$\bar{x}_T(k) = \bar{x}(kT)$$

Because e^{AT} is defined by (Reid, 1983: 254)

$$e^{AT} \equiv I + AT + \frac{A^2T^2}{2!} + \frac{A^3T^3}{3!} + \ldots$$

we have

$$\int_0^T e^{A\zeta}d\zeta = IT + \frac{AT^2}{2!} + \frac{A^2T^3}{3!} + \ldots$$

Therefore, we have

$$A_T = e^{AT} = I + AT + \frac{A^2T^2}{2!} + \frac{A^3T^3}{3!} + \ldots$$

$$B_T = \left[\int_0^T e^{A\xi} d\xi\right] B = \left[IT + \frac{AT^2}{2!} + \frac{A^2T^3}{3!} + \ldots\right] B$$

In order to define a different power series from which both A_T and B_T can be found, we can write

$$E \equiv IT + \frac{AT^2}{2!} + \frac{A^2T^3}{3!} + \ldots$$

This leaves

$$A_T = I + AE$$
$$B_T = EB$$

For the power series E to converge to an accurate result, a suitable combination of time increment T and matrix A must be sufficiently "small". With the "size" of the matrix A given by the Froebinius norm $||A_F||$, we want to find the smallest integer N such that

$$\frac{||A_F||T}{2^N} < 1 \qquad (4.5)$$

We will use this N in the Squaring Algorithm to calculate $e^{(AT/2^N)}$. From this, e^{AT} is found by squaring $e^{(AT/2^N)}$ a total of N times. The Squaring Algorithm for finding A_T and B_T is given as (Reid, 1983: 277):

1. Given the n by n matrix A, the n by 1 matrix

B, and the time increment T, find N such that Eq (4.5) is satisfied.

2. Calculate E_o from the power series

$$E_o \equiv I\frac{T}{2^N} + \frac{A\left(\frac{T}{2^N}\right)^2}{2!} + \frac{A^2\left(\frac{T}{2^N}\right)^3}{3!} + \ldots$$

Terminate this series when the last term is less than a specified tolerance.

3. Calculate F_o such that

$$F_o \equiv I + AE_o = e^{(AT/2^N)}$$

4. Now we square the term $e^{(AT/2^N)}$ N times to get e^{AT}. This is done with the recursion, M=1,2,3,...,N

$$E_M = E_{M-1} + F_{M-1}\,E_{M-1}$$
$$F_M = F_{M-1}F_{M-1}$$

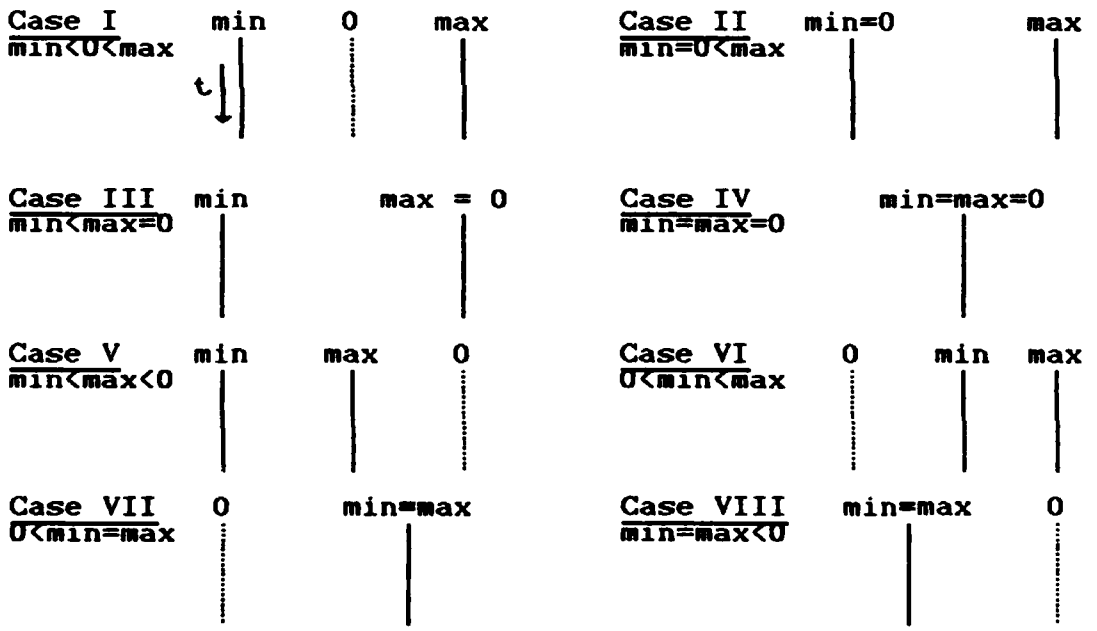
5. This leaves

$$A_T \equiv e^{AT} = F_N$$
$$B_T \equiv \left(\int_o^T e^{A\xi}d\xi\right)B = E_N B$$

This algorithm is performed in the subroutine SIMUL8. With A_T and B_T known, SIMUL8 calculates the time history of the system with Eq (4.4) for a step input. SIMUL8 records the time history of each of the N states in an n by k matrix, where k equals the number of discrete points calculated in SIMUL8. Row 1 contains the time history of state

1, Row 2 contains the time history of state 2, and so on. At this point, SIMUL8 passes the time history matrix to the subroutine which plots the histories, subroutine PLOT.

Plotting. When plotting the time history of a general system, we need to know the maximum and minimum values of the state to be plotted so as to scale the output for the screen. Because it would be convenient for the user to know when the state goes from a negative value to a positive value, or vice versa, we will plot the location of the value zero. We can see that, if we are to show the maximum, minimum, and zero values, the plotting routine must accomodate eight possible cases for a general system response.



Note that, because the subroutine SIMUL8 assumes initial conditions equal zero, cases V through VIII will not occur in MODES. However, to make PLOT flexible for other possible

applications, these cases have been accomodated.

By dividing the difference of the state x(t) and x(min) by the difference of x(max) and x(min), we have a relative "percentage" symbol "*" representing the state at time t. Here, x(t) is the value of the examined state at time t. The values of x(min) and x(max) are the minimum and maximum values of the variable throughout the time interval examined. Similarly, dividing the difference of zero and x(min) by the difference of x(max) and x(min), we find the location of the symbol "+" locating the zero value of the state. PLOT prints the maximum and minimum values of the state before and after plotting the time response for easy examination.

Finding the Resolvent Matrix and the Poles and Zeroes.

The Need for the Resolvent Matrix (Reid, 1983: 246). Given the general state-space model for an n^{th}-order, 1-input, m-output system

$$\dot{\bar{x}}(t) = A\bar{x}(t) + B\bar{u}(t) \qquad (4.6)$$

$$\bar{y}(t) = C\bar{x}(t) + D\bar{u}(t) \qquad (4.7)$$

with a given initial state $\bar{x}(0)$ at time t_o, we may want to examine the system's transfer function, H(s). With the transfer function, we can analyze the system using the classical, frequency-domain techniques.

To find the transfer function for this system, we first take the Laplace transform of Eq (4.6) and get

$$sX(s) - \bar{x}(0) = AX(s) + BU(s)$$

Solving for X(s) yields

$$X(s) = [sI-A]^{-1}\,\bar{x}(0) + [sI - A]^{-1}BU(s) \qquad (4.8)$$

Using Eqs (4.7) and (4.8), we have the Laplace transform of Eq (4.7) given as

$$Y(s) = C\left\{[sI-A]^{-1}\bar{x}(0) + [sI-A]^{-1}BU(s)\right\} + DU(s)$$

$$= C[sI-A]^{-1}\bar{x}(0) + \left\{C[sI-A]^{-1}B + D\right\}U(s) \qquad (4.9)$$

Separating this into the zero-input and zero-state portions (Reid, 1983:149), we get

$$Y_{zi}(s) = C[sI-A]^{-1}\bar{x}(0)$$

and $$Y_{zs}(s) = \left\{C[sI-A]^{-1}B + D\right\}U(s)$$

because the zero-state portion can be used to find the transfer function H(s) from the relation

$$Y_{zs}(s) = H(s)U(s)$$

we have the m by 1 transfer function given by

$$H(s) = C[sI-A]^{-1}B + D$$

Because the matrices A, B, C and D are all given in the general state-space model, only the n by n resolvent matrix $[sI-A]^{-1}$ must be found to facilitate using the classical analysis tools. The subroutine POLZER calculates the poles and zeroes by calling the subroutine RESOLV to calculate the

resolvent matrix. POLZER then postmultiplies this resolvent matrix by the input distribution matrix B. The subroutine PROOT is then used to find the roots of the polynomials in this transfer function matrix.

Calculating the Resolvent Matrix (Reid, 1983: 449-450). To invert [sI-A], where A is dimensioned n by n, we use

$$[sI-A]^{-1} = \frac{adj[sI-A]}{det[sI-A]} = \frac{Q(s)}{\Delta(s)} = \frac{Q_{n-1}s^{n-1}+ \ldots Q_1 s + Q_o}{s^n + b_{n-1} + \ldots b_1 s + b_o} \quad (4.10)$$

where the characteristic equation $\Delta(s)$ is defined by

$\Delta(s) \equiv det[sI-A] = s^n + b_{n-1}s^{n-1} + \ldots b_1 s_1 + b_o$.

Here the Q_i $(i=1,2,\ldots,n)$ are each n by n matrices.

The recursive algorithm used to calculate the matrices Q_i making up $Q(s)$ and the coefficients b_i making up $\Delta(s)$ is given by the following theorem attributed to Leverrier or Faddeev:

The matrices Q_i and coefficients b_i may be calculated recursively using

$$\begin{aligned}
Q_{n-1} &= I \\
b_{n-1} &= -\text{trace}\,(Q_{n-1}A) = -\text{trace}(A_1) \\
Q_{n-2} &= Q_{n-1}A + b_{n-1}I = A_1 + b_{n-1}I \\
b_{n-2} &= -\tfrac{1}{2}\,\text{trace}(Q_{n-2}A) = -\tfrac{1}{2}\text{trace}(A_2) \\
&\vdots \\
Q_1 &= Q_2A + b_2I = A_{n-2} + b_2I \\
b_1 &= -\tfrac{1}{n-1}\,\text{trace}(Q_1A) = -\tfrac{1}{n-1}\,\text{trace}(A_{n-1}) \\
Q_o &= Q_1A + b_1I = A_{n-1} + b_1I \\
b_o &= -\tfrac{1}{n}\,\text{trace}(Q_oA) = -\tfrac{1}{n}\,\text{trace}(A_n) \\
0 &= Q_oA + b_oI = A_n + b_oI
\end{aligned} \quad (4.11)$$

To prove this recursion finds the matrices Q_i, multiply both sides of Eq (4.10) by $\Delta(s)[sI-A]$ to give

$$\Delta(s)I = Q(s)[sI-A] \qquad (4.12)$$

By substituting our expressions for $\Delta(s)$ and $Q(s)$ into Eq (4.12), we get

$$(s^n + b_{n-1}s^{n-1} + \ldots + b_1 s + b_o)I = Q_{n-1}s^{n-1}s - Q_{n-1}s^{n-1}A) + \ldots + Q_1 ss - Q_1 sA + Q_o s - Q_o A$$

Collecting similar terms of s for the right-hand side of this equation, we get

$$(s^n + b_{n-1}s^{n-1} + \ldots + b_1 s + b_o)I = Q_{n-1}s^n + (Q_{n-2} - Q_{n-1}A)s^{n-1} + \ldots + (Q_1 - Q_2 A)s^2 + (Q_o - Q_1 A)s - Q_o A$$

Equating like powers of s, we have

$$\begin{aligned} I &= Q_{n-1} \\ b_{n-1}I &= Q_{n-2} - Q_{n-1}A \\ &\vdots \\ b_2 I &= Q_1 - Q_2 A \\ b_1 I &= Q_o - Q_1 A \\ b_o I &= Q_o A \end{aligned}$$

Rearranging the terms of these equations give the previous equations for this recursive routine.

Before we show that this recursion finds the coefficiencts b_i, we must introduce some tools. The first of these is known as Newton's Formulas (Uspensky, 1948: 261).

Newton's Formulas relate the coefficients of a polynomial

$$f(x) = (x-\lambda_1)(x-\lambda_2)\ldots(x-\lambda_n) = x^n+p_1x^{n-1} +\ldots+p_n$$

to the sum of the powers

$$\lambda_1^{\alpha} + \lambda_2^{\alpha} + \ldots + \lambda_n^{\alpha}$$

Using the notation s_α to represent this sum of powers, Newton's Formulas state that

$$s_k + p_1 s_{k-1} + \ldots + kp_k = 0 \quad (k=1,2,\ldots,n-1)$$

For our representation of the characteristic equation, Newton's Formulas are given as

$$s_k + b_{n-1}s_{k-1} + \ldots + b_{n-(k-2)}s_2 + b_{n-(k-1)}s_1 + kb_{n-k} = 0 \quad (k=1,2,\ldots,n-1) \qquad (4.13)$$

Next, we will need the following theorem (Gantmacher, 1959: 84):

If $\lambda_1, \lambda_2,\ldots,\lambda_n$ are the eigenvalues of a matrix A, and if $g(\mu)$ is a scalar polynomial, then $g(\lambda_1)$, $g(\lambda_2)$, $g(\lambda_3),\ldots, g(\lambda_n)$ are the eigenvalues of $g(A)$. With this, the k^{th} power of A, A^k, has eigenvalues $\lambda_1^k, \lambda_2^k,\ldots\lambda_n^k$. Because $\text{trace}(A) = \lambda_1+ \lambda_2+\ldots\lambda_n = a_{11} + a_{22} + \ldots + a_{nn}$, where a_{ij} is the element of A in the i^{th} row and j^{th} column, we have

$$\text{trace}(A^k) = \sum_{i=1}^{n} \lambda_i^k \qquad (k = 0, 1, 2,\ldots n)$$

Note that trace(A^k) equals the s_k from Newton's Formulas. With these tools, we will now show that the Leverrier or Faddeev recursive algorithm does find the required coefficients b_i (Gantmacher, 1959: 87).

Consider the expressions for A_k, $k = 1, 2, \ldots, n$. We have

$$A_1 = Q_{n-1}A = IA = A$$

$$A_2 = (A_1 + b_{n-1} I)A = A^2 + b_{n-1}A$$

and

$$A_3 = (A_2 + b_{n-2}I)A = (A^2 + b_{n-1}A + b_{n-2}I)A = A^3 + b_{n-1}A^2 + b_{n-2}A$$

Extending this to A_k, we see that

$$A_k = A^k + b_{n-1}A^{k-1} + \ldots + b_{n-(k-1)}A \qquad (4.14)$$

Now we will equate the trace of the left-hand side of Eq (4.14) to the trace of the right-hand side

$$\text{trace}(A_k) = s_k + b_{n-1}s_{k-1} + \ldots + b_{n-k+2}s_2 + b_{n-k+1}s_1 \qquad (4.15)$$

From the recursion formulas, Eqs (4.11), we have

$$b_{n-k} = -\frac{1}{k}\,\text{trace}(A_k)$$

This gives $\text{trace}(A_k) = -k\,b_{n-k}$. Substituting this into Eq (4.15) gives

$$-kb_{n-k} = s_k + b_{n-1}s_{k-1} + \ldots + b_{n-(k-2)}s_2 + b_{n-(k-1)}s_1$$
$$(k = 1,2,\ldots,n)$$

These formulas are Newton's formulas, Eq (4.13), by which the coefficients of the characteristic polynomial $\Delta(s)$ are determined successively. Therefore, the given recursion does find the coefficients of the characteristic equation.

Note that because this algorithm is known to be numerically sensitive and prone to wordlength-related computational errors (Reid, 1983:450), and because the problem of calculating the zeroes of polynomials often is sensitive to small changes in the coefficients of the polynomials (Williams, 1986: 73), the Leverrier or Faddeev recursion is implemented using Fortran's double precision facilities.

Finding the Roots of a Polynomial (Hamming, 1962: 356-359). Our remaining task is to find the poles and zeroes of the product of the resolvent matrix and the input distribution matrix. We will use the Bairstow method to find these roots. The Bairstow method works for polynomials of degree 3 or greater

Let the polynomial of interest be

$$P(x) = a_0 + a_1x + a_2x^2 + \quad + a_nx^n$$

We will assume we have a guess at a quadratic factor for this polynomial of

$$x^2 + px + q$$

Using synthetic division, we will divide the polynomial by this assumed quadratic factor to get a quotient and a

remainder.

$$a_n x^n + a_{n-1}x^{n-1} + \ldots + a_o =$$
$$(x^2 + px + q)(b_n x^{n-2} + b_{n-1}x^{n-3} + \ldots + b_2) + b_1 x + b_o$$

Equating like powers of x, we find the coefficients b_i to be

$$\begin{aligned}
b_n &= a_n \\
b_{n-1} &= a_{n-1} - pb_n \\
b_{n-2} &= a_{n-2} - pb_{n-1} - qb_n \\
&\vdots \\
b_{n-k} &= a_{n-k} - pb_{n-k+1} - qb_{n-k+2} \\
&\qquad\qquad (k = 2, 3, \ldots, n-1) \\
&\vdots \\
b_o &= a_o - qb_2
\end{aligned} \tag{4.16}$$

If the remainder of the synthetic division equals zero, that is, if

$$b_o = b_1 = 0$$

then the quadratic factor used is in fact the quadratic factor we want. When the remainder does not equal zero, we must adjust our guess for the quadratic factor. To do this, we will assume the remainder coefficients b_1 and b_o to be functions of the assumed quadratic coefficients p and q.

$$b_1 = b_1(p,q)$$
$$b_o = b_o(p,q)$$

Now we will use the two-dimensional analog to Newton's

method and use a Taylor series to expand b_1 and b_0 about the current guess (p,q). Representing the desired solution as p^* and q^*, we have

$$b_1(p^*,q^*) = 0 = b_1(p,q) + \frac{\partial b_1}{\partial p}\Delta p + \frac{\partial b_1}{\partial q}\Delta q + \ldots \quad (4.17)$$

$$b_0(p^*,q^*) = 0 = b_0(p,q) + \frac{\partial b_0}{\partial p}\Delta p + \frac{\partial b_0}{\partial q}\Delta q + \ldots \quad (4.18)$$

where $\Delta p = p^* - p$

$\Delta q = q^* - q$

Δp and Δq are the errors to be corrected for our next guess. Using only the first order terms from these Taylor series, we have two linear equations for the changes to be made in p and q. We will analytically find the partial derivatives for our Taylor series.

Differentiating Eqs (4.16) with respect to p, we get

$$\frac{\partial b_n}{\partial p} = 0$$

$$\frac{\partial b_{n-1}}{\partial p} = -b_n - p\frac{\partial b_n}{\partial p}$$

$$\frac{\partial b_{n-2}}{\partial p} = -b_{n-1} - p\frac{\partial b_{n-1}}{\partial p} - q\frac{\partial b_n}{\partial p}$$

$$\vdots$$

$$\frac{\partial b_{n-k}}{\partial p} = -b_{n\ k+1} - p\frac{\partial b_{n-k+1}}{\partial p} - q\frac{\partial b_{n-k+2}}{\partial p}$$

$$\vdots$$

$$\frac{\partial b_0}{\partial p} = -q\frac{\partial b_2}{\partial p}$$

If we use the notation

$$\frac{\partial b_k}{\partial p} = -C_k^* \tag{4.19}$$

then we can write

$$\begin{aligned} C_n^* &= 0 \\ C_{n-1}^* &= b_n - pC_n^* \\ C_{n-2}^* &= b_{n-1} - pC_{n-1}^* - qC_n^* \\ &\vdots \\ C_{n-k}^* &= b_{n-k+1} - pC_{n-k+1}^* - qC_{n-k+2}^* \\ &\vdots \\ C_0^* &= -qC_2^* \end{aligned} \tag{4.20}$$

If we identify the term b_k of Eq (4.20) with a_{k-1} of Eq (4.16), and C_{n-k}^* with b_{n-k+1}, and if we excuse the difference in the last equation, then Eqs (4.16) and (4.20) are of the same form. We will now repeat the synthetic division using the same quadratic factor $x^2 + px + q$, and perform the division on the polynomial

$$b_0 + b_1 x + \ldots + b_n x^n$$

Similar to Eqs (4.16), this gives us

$$\begin{aligned} C_n &= b_n \\ C_{n-1} &= b_{n-1} - pC_n \\ C_{n-2} &= b_{n-2} - pC_{n-1} - qC_n \end{aligned}$$

$$\vdots$$

$$C_{n-k} = b_{n-k} - pC_{n-k+1} - qC_{n-k+2}$$

$$\vdots$$

$$C_1 = b_1 - pC_2 - qC_3$$

$$C_0 = b_0 - qC_2 \qquad (4.21)$$

Using the notation in (4.19), the partial derivatives with respect to p we are looking for in Eqs (4.17) and (4.18) are

$$\frac{\partial b_1}{\partial p} = -C_1^* \qquad \frac{\partial b_0}{\partial p} = -C_0^*$$

Comparing Eqs (4.20) and (4.21) we see that

$$C_{k-1}^* = C_k \qquad (k = n, n-1, \ldots, 3, 2)$$

$$C_0^* = C_1 - b_1 + pC_2$$

Therefore

$$\frac{\partial b_1}{\partial p} = -C_1^* = -C_2$$

$$\frac{\partial b_0}{\partial p} = -C_0^* = -(C_1 - b_1 + pC_2)$$

Now we must find expressions for the partial derivatives with respect to q. As before, we will differentiate Eqs (4.16) to get

$$\frac{\partial b_{n-1}}{\partial q} - p\frac{\partial b_n}{\partial q} = 0$$

$$\frac{\partial b_{n-2}}{\partial q} = -b_n - p\frac{\partial b_{n-1}}{\partial q} - q\frac{\partial b_n}{\partial q}$$

$$\vdots$$

$$\frac{\partial b_{n-k}}{\partial q} = -b_{n-k+2} - p\frac{\partial b_{n-k+1}}{\partial q} - q\frac{\partial b_{n-k+2}}{\partial q}$$

$$\vdots$$

$$\frac{\partial b_0}{\partial q} = -b_2 - q\frac{\partial b_2}{\partial q}$$

Using the notation

$$\frac{\partial b_k}{\partial q} = -C_k^{\bullet\bullet}$$

we have

$$C_{n-1}^{\bullet\bullet} = 0$$

$$C_{n-2}^{\bullet\bullet} = b_n - pC_{n-1}^{\bullet\bullet} - qC_n^{\bullet\bullet}$$

$$\vdots$$

$$C_{n-k}^{\bullet\bullet} = b_{n-k+2} - pC_{n-k+1}^{\bullet\bullet} - qC_{n-k+2}^{\bullet\bullet}$$

$$\vdots$$

$$C_0^{\bullet\bullet} = b_2 - qC_2^{\bullet\bullet} \qquad (4.22)$$

Because $C_n^{\bullet\bullet} = C_{n+1}^{\bullet\bullet} = 0$, we can use the relations

$$C_{k+2}^{\bullet\bullet} = C_k \quad (k = n, n-1, \ldots, 3)$$

$$C_0^{\bullet\bullet} = C_2 + pC_3$$

to compare Eqs (4.21) and (4.22). With these transformations, Eqs (4.21) and (4.22) are the same. Therefore, the partial derivatives with respect to q are

$$\frac{\partial b_1}{\partial q} = -C_1^{**} = -C_3$$

$$\frac{\partial b_0}{\partial q} = -C_0^{**} = -(C_2 + pC_3)$$

We now have all that is needed to find the amount to change our guess at the quadratic factor $x^2 + px + q$. Eqs (4.17) and (4.18) become

$$b_1(p,q) = C_2\Delta p + C_3\Delta q$$

$$b_0(p,q) = (C_1 - b_1 + pC_2)\Delta p + (C_2 + pC_3)\Delta q$$

Solving these two equations yield the required changes in p and q. In this way, we have an iterative approach to find a quadratic factor of the original polynomial. When we find a quadratic factor, we can factor it out and use the remaining quotient as a new polynomial to examine in the same fashion. The subroutine PROOT using the Bairstow method (Melsa and Jones, 1973: 161-163) completes our effort to examine the system's transfer function H(s).

Inverting a Matrix.

The subroutines INVERT, FACTOR, and SUBST are used to invert a square matrix. These subroutines use the Gauss-Jordan elimination method with row pivoting to calculate accurately the inverse matrix. The details of this procedure are presented below.

The inverse of a matrix C is defined to be C^{-1}, such that

$$CC^{-1} = I \qquad (4.23)$$

where I is the identity matrix. If we apply a series of row operations to the C matrix on the left-hand side of this equation to transform C into I, we would have to apply the same row operations to the right-hand side to preserve the equality. This would give us

$$IC^{-1} = C^{-1}$$

Therefore, the series of row operations which transform the C matrix into the identity matrix will also transform the identity matrix into C^{-1}. The series of row operations used for these transformations are known as Gauss-Jordan elimination.

Gauss-Jordan elimination uses two types of operations to transform a matrix into a different form. These are (Hornbeck, 1975: 92)

1) Multiplication or division of a row by a constant.
2) Replacement of any row by the sum (or difference) of that row and any other row.

When we consider the elements of our matrices as representing coefficients of linear, independent equations, we can see that these operations do not affect the equality of Eq (4.23).

Therefore, our task is to use the above two types of operations to transform C into I, and by applying the same operations on I, transform I into C^{-1}. This method is best understood with an example (Hornbeck, 1975: 99). We will examine the 3 by 3 matrix

$$C = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

Writing C and I together, we will perform our row operations on both matrices at the same time.

$$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To transform the left matrix into the identity matrix, the first element must become 1. To do this, we divide the first row of both matrices by 2:

$$\begin{bmatrix} 1 & 1/2 & 1/2 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \quad \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now we want the first column of the left matrix to become

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

We can achieve this by subtracting the first row from the second row and replacing the second row with this difference, and by subtracting the first row from the third row and replacing the third row with this difference:

$$\begin{bmatrix} 1 & 1/2 & 1/2 \\ 0 & 3/2 & 1/2 \\ 0 & 1/2 & 3/2 \end{bmatrix} \quad \begin{bmatrix} 1/2 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/2 & 0 & 1 \end{bmatrix}$$

Here, the first column of the left matrix is as we want it. Our attention now shifts to the second column.

Using a similar approach, we want to transform the second element in the second column into 1. We do this by dividing the second row of both matrices by the "pivot element", that is, the number which is the position where we want the 1. Here, the pivot element is 3/2. This divison gives

$$\begin{bmatrix} 1 & 1/2 & 1/2 \\ 0 & 1 & 1/3 \\ 0 & 1/2 & 3/2 \end{bmatrix} \quad \begin{bmatrix} 1/2 & 0 & 0 \\ -1/3 & 2/3 & 0 \\ -1/2 & 0 & 1 \end{bmatrix}$$

To transform the first element of the second column of the left matrix into zero, we multiply the second row by 1/2, subtract it from the first row, and replace the first row with this difference. A similar procedure will also transform the last element of the second column into zero.

$$\begin{bmatrix} 1 & 0 & 1/3 \\ 0 & 1 & 1/3 \\ 0 & 0 & 4/3 \end{bmatrix} \quad \begin{bmatrix} 2/3 & -1/3 & 0 \\ -1/3 & 2/3 & 0 \\ -1/3 & -1/3 & 1 \end{bmatrix}$$

Now we must apply this method to the last column. Repeating the above procedure, we finaly get

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 3/4 & -1/4 & -1/4 \\ -1/4 & 3/4 & -1/4 \\ -1/4 & -1/4 & 3/4 \end{bmatrix}$$

With the left matrix transformed into the identity matrix, the right matrix is now C^{-1}. To verify this, CC^{-1} is

$$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} 3/4 & -1/4 & -1/4 \\ -1/4 & 3/4 & -1/4 \\ -1/4 & -1/4 & 3/4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I$$

Since C^{-1} is defined by $CC^{-1} = I$, we have found C^{-1}.

One computational problem we can have with the Gauss-Jordan elimination method involves the magnitude of the pivot element (Hornbeck, 1975: 96). With the various computations performed, any given element can become very small in magnitude compared to the rest of the elements in its row, as well as being inaccurate. When we divide the other elements of that row by the pivot element, we may introduce unacceptable roundoff errors, resulting in poor results. By interchanging columns of the matrix so as to shift the largest element (in magnitude) in the row of interest to the pivot position, we can reduce this problem. Every column interchange in the left matrix requires an equivalent column interchange in the right matrix.

Utility Routines

MODES requires several subroutines which perform simple, repetitive tasks. Classifying these tasks as utility routines, we will list each of these subroutines

here, and briefly discuss their use. Because they perform common tasks, further discussions are not warranted here.

AXBEQC. This is used to multiply two matrices (A times B equals C).

DAXBEC. This is a double precision routine used to multiply two matrices.

DIMENA. This asks the user for the dimensions of the A and B matrices for the equation $\dot{\bar{x}} = A\bar{x} + B\bar{u}$.

INPUTA. After DIMENA defines the dimensions, this is used to enter the A and B matrices. The user may either enter a matrix directly from the keyboard or read a matrix from an existing computer file.

PRINTA. Used to print a real matrix.

PRNVEC. This routine prints a vector.

SAVE. When we save data to a computer file, SAVE asks for the required information and saves the data.

YESORN. Whenever the user is asked a yes or no question, YESORN is used to retrieve the answer.

V. Users' Guide to the Subroutines of MODES

This manual is a guide to all of the subroutines used by the program MODES. The theory behind these routines are presented in previous chapters, as needed.

Except as noted, all input variables to each subroutine remain unchanged on output.

In order to compile MODES for the IBM PC compatible computers using version 3.2 of the Microsoft FORTRAN77 compiler, some changes are required to bring the program within the memory limits of the compiler. Specifically, the matrices must be dimensioned for 10 states instead of the current 15. COMMON blocks are also helpful. Specifically, the 10 by 10 by 10 matrices in RESOLV and POLZER, as well as the 10 by 151 matrix XHISTR in SIMUL8 and PLOT should be placed in COMMON blocks. Also, the main program should have some additional subroutines taken from it. The steps within the DO-loop were selected. Another change, not required for memory considerations, is for the subroutine SAVE. The OPEN statement between lines 40 and 50 must have STATUS = "OLD". The Microsoft Compiler version 3.2 does not recognize "UNKNOWN"

Subroutine ATLDEF

Purpose

Calculates the closed-loop system matrix $\tilde{A}$ from the matrices A, B and G such that

$$\tilde{A} = A + BG$$

A and B are defined for the linear time-invariant state equation

$$\dot{\bar{x}} = A\bar{x} + B\bar{u}$$

and G is the feedback gain matrix required to optimize the eigenstructure, and therefore the time response characteristics, of the modified system

$$\dot{\bar{x}} = \tilde{A}\bar{x}$$

Calling Sequence:

CALL ATLDEF (A,B,G,ATILDE,NMAX,N,M,DIARY)

Input Variables:

A	The real N by N open-loop system matrix.
B	The real N by M input matrix.
G	The real M by N feedback gain matrix.
NMAX	The integer number of rows used in the original DIMENSION statement for the matrices A, B, G and ATILDE in the calling program. Note: the original DIMENSION statement for each of these

matrices must have the same number of rows.

N — The integer number of rows of the A and B matrices, and the number of columns of the A and G matrices. N must not be greater than NMAX.

M — The integer number of rows of the G matrix and the number of columns of the B matrix. M must not be greater than N.

DIARY — The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

ATILDE — The real N by N closed-loop matrix $\tilde{A}$.

Subroutine ATRANS

Purpose:

Defines the transpose of a matrix.

Calling Sequence:

CALL ATRANS (NMAX,MMAX,N,M,A,AT)

Input Variables:

NMAX	The integer number of rows used in the original DIMENSION statement for the matrix A in the calling program.
MMAX	The integer number of rows used in the original DIMENSION statement for the matrix AT in the calling program.
N	The integer number of rows of the matrix A and the number of columns of the matrix A^T. N must not be greater than NMAX.
M	The integer number of columns of the matrix A and the number of rows of the matrix A^T. M must not be greater than MMAX.
A	The real N by M matrix whose transpose is to be found.

Output Variables:

AT	The real M by N transpose of the input matrix A.

Subroutine AXBEQC

Purpose:

Multiplies two matrices to yield a third matrix. AB = C .

Calling Sequence:

CALL AXBEQC (A,NMAXA,NA,MA,B,NMAXB,NB,MB,C,NMAXC,DIARY)

Input Variables:

A	The real NA by MA first matrix in the equation AB = C .
NMAXA	The integer number of rows used in the original DIMENSION statement for the matrix A in the calling program.
NA	The integer number of rows of the matrix A. NA must not be greater than NMAXA.
MA	The integer number of columns of the matrix A.
B	The real NB by MB matrix in the equation AB = C .
NMAXB	The integer number of rows used in the original DIMENSION statement for the matrix B in the calling program.
NB	The integer number of rows of the matrix B. NB must not be greater than NMAXB.
MB	The integer number of columns of the matrix B.
NMAXC	The integer number of rows used in the original DIMENSION statement for the matrix C in the calling program.
DIARY	The logical variable indicating if any WRITE

statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

C The real NA by MB matrix product of the matrices A and B.

If the matrices to be multiplied have incompatible dimensions, an error statement is printed.

Subroutine BALANC

Purpose

Balances a real general matrix to prepare the matrix so that a more accurate calculation of the matrix eigenstructure can be made.

Calling Sequence:

CALL BALANC (NM,N,A,LOW,IGH,SCALE)

Input Variables:

NM	The integer number of rows used in the original DIMENSION statement for the matrix whose eigenvalues are desired, the matrix A.
N	The integer number of rows of the square matrix A. N must not be greater than NM.
A	The real N by N matrix whose eigenstructure is desired. On output, A is altered. Therefore, if the original A matrix will be needed after calling BALANC, one must use a duplicate matrix in the calling statement.

Output Variables:

A	The real N by N balanced matrix whose eigenvalues equal the eigenvalues of the input A matrix.
LOW, IGH	Integer variables indicating the boundary indices for the balanced matrix. These variables are to be used by other eigenstructure routines.

SCALE — A real one-dimensional matrix containing information about the similarity transformations used to balance the matrix. SCALE at least must be of dimension N. This information is to be used when transforming the calculated eigenvectors of the balanced matrix into the eigenvectors of the original matrix.

Associated Subroutines:

BALBAK, EIGV, ELMHES, ELTRAN and HQR2

References:

1. Smith, B.T., et al. Matrix Eigensystem Routines-EISPACK Guide. Lecture Notes in Computer Science, Volume 6 (Second Edition), edited by G. Goos and J. Hartmanis. New York: Springer-Verlag: 200-203 (1976).

2. Parlett, B.N., and C. Reinsch. "Balancing a Matrix for Calculation of Eigenvalues and Eigenvectors," Numerische Mathematik, 13: 293-304 (August 1969).

Subroutine BALBAK

Purpose:

Transforms the eigenvectors of a matrix balanced by the subroutine BALANC into the real eigenvectors of the original matrix. See subroutine BALANC.

Calling Sequence:

CALL BALBAK (NM,N,LOW,IGH,SCALE,M,Z)

Input Variables:

NM	The integer number of rows used in the original DIMENSION statement for the matrix Z in the calling program.
N	The integer equal to the number of components of the vectors in the matrix Z. N must not be greater than NM.
LOW,IGH	Integer variables indicating the boundary indices for the balanced matrix.
SCALE	A real one-dimensional matrix containing information about the similarity transformations used to balance the original matrix. SCALE at least must be of dimension N.
M	An integer variable set equal to the number of columns of Z to be back-transformed.
Z	A real two-dimensional matrix. The first M columns contain the real and imaginary parts of the eigenvectors to be back-transformed.

Output Variables:

Z — The real N by N matrix of eigenvectors of the original real, general matrix. The first M columns contain the real and imaginary components of the eigenvectors. If the i^{th} eigenvalue is real, then the i^{th} column of Z contains the corresponding real eigenvector. If the i^{th} eigenvalue is the first of a pair of complex conjugate eigenvalues, then the i^{th} column of Z contains the real components of the associated complex eigenvector, and the $i+1^{st}$ column contains the imaginary components of this eigenvector.

Associated Subroutines:

BALANC, EIGV, ELMHES, ELTRAN and HQR2

References:

1. Smith, B.T., et al. Matrix Eigensystem Routines-EISPACK Guide. Lecture Notes in Computer Science, Volume 6 (Second Edition), edited by G. Goos and J. Hartmanis. New York: Springer-Verlag: 200-203 (1976).

2. Parlett, B.N., and C. Reinsch. "Balancing a Matrix for Calculation of Eigenvalues and Eigenvectors," Numerische Mathematik, 13: 293-304 (August 1969).

Subroutine COPYAB

Purpose:

Copies one matrix, A, and names the duplicate matrix B.

Calling Sequence:

CALL COPYAB (NMAX,N,M,A,B)

Input Variables:

NMAX	The integer number of rows used in the original DIMENSION statement for the matrices A and B in the calling program. The original DIMENSION statement for both matrices must have the same number of rows.
N	The integer number of rows of the matrices A and B.
M	The integer number of columns of the matrices A and B.
A	The real N by M matrix to be copied.

Output Variables:

B	The real N by M matrix which equals the input matrix A.

Subroutine DAXBEC

Purpose:

Multiplies two matrices to yield a third matrix. AB = C DAXBEC is the DOUBLE PRECISION equivalent of the subroutine AXBEQC.

Calling Sequence:

CALL DAXBEC (A,NMAXA,NA,MA,B,NMAXB,NB,MB,C,NMAXC,DIARY)

Input Variables:

A	The real NA by MA first matrix in the equation AB = C .
NMAXA	The integer number of rows used in the original DIMENSION statement for the matrix A in the calling program.
NA	The integer number of rows of the matrix A. NA must not be greater than NMAXA. Also, NA must not be greater than NMAXC.
MA	The integer number of columns of the matrix A.
B	The real NB by NB matrix in the equation AB = C .
NMAXB	The integer number of rows used in the original DIMENSION statement for the matrix B in the calling program.
NB	The integer number of rows of the matrix B. NB must not be greater than NMAXB.
MB	The integer number of columns of the matrix B.
NMAXC	The integer number of rows used in the original

DIMENSION statement for the matrix C in the calling program

DIARY — The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = TRUE, then the WRITE statement will be made. If DIARY = FALSE, the WRITE statement will not be made.

Output Variables:

C — The real NA by MB product of the matrices A and B.

If the matrices to be multiplied have incompatible dimensions, an error statement is printed.

Subroutine DIMENA

Purpose:

Asks the user for the dimensions of a two-dimensional matrix which is to be defined.

Calling Sequence:

Call DIMENA (N,M,DIARY)

Input Variables:

DIARY — The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

N — The integer number of rows of the matrix to be defined.

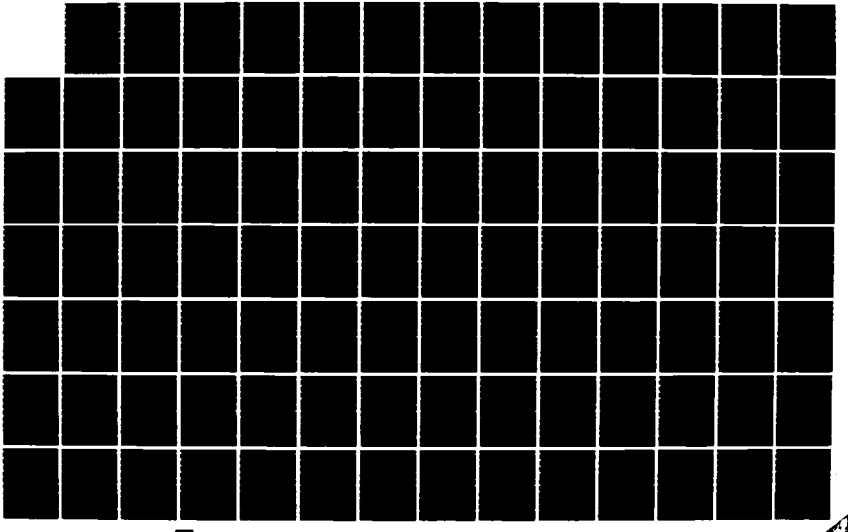
M — The integer number of columns of the matrix to be defined.

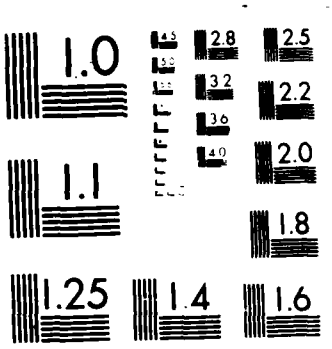
Subroutines Called:

YESORN

AD-A179 341 MULTI-INPUT/MULTI-OUTPUT DESIGNATED EIGENSTRUCTURE (MODES): A COMPUTER-AI... (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... M E HOPPER

UNCLASSIFIED MAR 87 AFIT/GAE/AA/87M-2 F/G 12/2 NL





Subroutine EIGSTR

Purpose:

Asks the user for the desired eigenvalues and eigenvectors, and defines the desired eigenstructure matrices from that information.

Calling Sequence:

CALL EIGSTR (EVEC,EVAL,NMX2,N,DIARY)

Input Variables:

NMX2 The integer number of rows used in the original DIMENSION statement for the matrix EVAL in the calling program. Also the dimension used in the original DIMENSION statement for the one-dimensional matrix EVAL. NMX2 must be at least twice the number of eigenvalues to be defined.

N The integer number of eigenvalues to be defined.

DIARY The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

EVEC The real 2N by N matrix of eigenvectors. The eigenvectors are stored in the columns of EVEC. The first N elements of any column are the real

components of the eigenvector; the second N elements of any column are the imaginary components.

EVAL — The real 2N by 1 matrix of eigenvalues. The i^{th} element ($i \leq N$) is the real component of the i^{th} eigenvalue; the $i+N^{th}$ element is the imaginary component. The i^{th} eigenvalue is associated with the eigenvector in the i^{th} column of EVEC.

Subroutines Used:

YESORN

Subroutine EIGV

Purpose:

Calculates the eigenvalues and eigenvectors of a general, real matrix.

Calling Sequence:

CALL EIGV (A,Z,WR,WI,NMAX,N,DIARY)

Input Variables:

A	The real N by N matrix whose eigenstructure is desired. Restriction on N: N≤NMAX≤15.
NMAX	The integer number of rows used in the original DIMENSION statement for the matrices A and Z in the calling program. The original DIMENSION statements for both A and Z must have the same number of rows. Restriction on NMAX: NMAX≤15.
N	The integer number of rows of the square matrices A and Z. Also, the number of elements in the one-dimensional vectors WR and WI. N must not be greater than NMAX.
DIARY	The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

Z — The real N by N matrix of eigenvectors. When the i^{th} eigenvalue is real (WI(i)=0), the i^{th} column contains the components of the associated real eigenvector. When the i^{th} eigenvalue is the first of a pair of complex conjugate eigenvalues, the i^{th} column contains the real components of the associated complex eigenvector; the $i+1^{st}$ column contains the imaginary components.

WR — The real vector, of dimension N, containing the real components of the eigenvalues of the matrix A. The i^{th} eigenvalue corresponds to the i^{th} eigenvector in Z.

WI — The real vector, of dimension N, containing the imaginary components of the eigenvalues of matrix A. The i^{th} eigenvalue corresponds to the i^{th} eigenvector in Z.

Subroutines Used:

BALANC, BALBAK, COPYAB, ELMHES, ELTRAN, and HQR2

Subroutine ELMHES

Purpose:

Reduces a real, general matrix to upper Hessenberg form so that subroutine HQR2 can calculate the eigenstruture of the original matrix.

Calling Sequence:

CALL ELMHES (NM,N,LOW,IGH,A,INT)

Input Variables:

NM	The integer number of rows used in the original DIMENSION statement for the matrix A in the calling program.
N	The integer number of rows of the square matrix A. N must not be greater than NM.
LOW,IGH	The integer variables indicating the boundary indices for the balanced matrix. If the matrix is not balanced, set LOW=1 and IGH=N. See subroutine BALANC.
A	The real N by N matrix to be transformed to upper Hessenberg form.

Output Variables:

A	The upper Hessenberg matrix and the multipliers used to perform the reduction.
INT	An integer one-dimensional variable of dimension at least IGH which identifies the rows and columns

interchanged during the reduction.

Associated Subroutines:

BALANC, BALBAK, EIGV, ELTRAN and HQR2

References:

1. Smith, B.T., et al. Matrix Eigensystem Routines-EISPACK Guide. Lecture Notes in Computer Science, Volume 6 (Second Edition), edited by G. Goos and J. Hartmanis. New York: Springer-Verlag: 200-203 (1976).

2. Martin, R.S., and J.H. Wilkinson. "Similarity Reduction of a General Matrix to Hessenberg Form," Numerische Mathematik, 12: 349-368 (December 1968).

Subroutine ELTRAN

Purpose:

Accumulates the similarity transformations used in the reduction of a real general matrix to upper Hessenberg form by ELMHES.

Calling Sequence:

CALL ELTRAN (NM,N,LOW,IGH,A,INT,Z)

Input Variables:

NM	The integer number of rows used in the original DIMENSION statement for the matrices A and Z in the calling program. The original DIMENSION statements for both A and Z must have the same number of rows.
N	The integer number of rows of the square matrix A. N must not be greater than NM.
LOW,IGH	The integer input variables indicating the boundary indices for the balanced matrix. If the matrix is not balanced, set LOW=1 and IGH=N.
A	The real two-dimensional matrix containing the multipliers which were used in the reduction to the Hessenberg form in its lower triangle below the subdiagonal.
INT	The one-dimensional integer variables identifying the rows and columns interchanged during the reduction by ELMHES.

Output Variables:

Z — The real two-dimensional transformation matrix produced in the reduction to the upper Hessenberg form by ELMHES.

Associated Subroutines:

BALANC, BALBAK, EIGV, ELMHES, and HQR2

References:

1. Smith, B.T., et al. Matrix Eigensystem Routines-EISPACK Guide. Lecture Notes in Computer Science, Volume 6 (Second Edition), edited by G. Goos and J. Hartmanis. New York: Springer-Verlag: 200-203 (1976).

2. Peters, G., and J.H. Wilkinson. "Eigenvectors of Real and Complex Matrices By LR and QR Triangularizations," Numerische Mathematik, 16: 181-204 (1970).

Subroutine FACTOR

Purpose:

Performs a factorization of a square matrix as the first step of the Gauss-Jordan elimination method to calculate the inverse of a matrix.

Calling Sequence:

CALL FACTOR (A,W,IPIVOT,D,N,IFLAG,NM)

Input Variables:

A	The real N by N matrix to be inverted.
W	A real N by N working array.
N	The integer number of rows for both square matrices A and W.
NM	The integer number of rows used in the original DIMENSION statement for the matrices A and W. The original DIMENSION statement for both matrices must have the same number of rows.

Output Variables:

W	The real N by N working array.
IPIVOT	The integer vector of dimension N identifying the pivot elements for the Gauss-Jordan elimination.
D	The real vector of dimension N storing the absolute value of the element of each row of the matrix A which has the largest magnitude of all elements in that row.

IFLAG The integer variable used to identify when the A matrix is singular.

Associated Subroutines:

INVERT and SUBST

Subroutine HQR2

Purpose:

Computes the eigenvalues and eigenvectors of a real upper Hessenberg matrix using the QR method.

Calling Sequence:

CALL HQR2 (NM,N,LOW,IGH,H,WR,WI,Z,IERR)

Input Variables:

NM The integer number of rows used in the original DIMENSION statement for the matrices H and Z. The original DIMENSION statement for both matrices must have the same number of rows.

N The number of rows in both square matrices A and Z. N must not be greater than NM.

LOW,IGH The integer variables indicating the boundary indices for the balanced matrix. If the matrix is not balanced, set LOW=1 and IGH=N.

H The real N by N upper Hessenberg matrix. Note: HQR2 destroys this matrix on output.

Z If the eigenvectors of the upper Hessenberg matrix

are desired, Z contains the identity matrix of order N. If the eigenvectors of a real general matrix are desired, then Z is the transformation matrix produced in ELTRAN which reduced the general matrix to Hessenberg form.

Output Variables:

WR — The real one-dimensional array of dimension N containing the real parts of the eigenvalues.

WI — The real one-dimensional array of dimension N containing the imaginary parts of the eigenvalues.

Z — The real N by N matrix of eigenvectors. If the i^{th} eigenvalue is real, then the i^{th} column of Z contains the associated eigenvector. If the i^{th} eigenvalue is the first of a pair of complex conjugate eigenvalues, then the i^{th} column of Z is the real part of the associated complex eigenvector, and the $i+1^{st}$ column is the imaginary part of this eigenvector.

IERR — The integer error code. If IERR=0 then HQR2 has converged to an answer. If IERR≠0, then HQR2 has not converged.

Associated Subroutines:

BALANC, BALBAK, EIGV, ELMHES, ELTRAN

References:

1. Smith, B.T., et al. Matrix Eigensystem Routines-EISPACK Guide. Lecture Notes in Computer Science, Volume 6 (Second Edition), edited by G. Goos and J. Hartmanis. New York: Springer-Verlag: 200-203 (1976).

2. Peters, G., and J.H. Wilkinson. "Eigenvectors of Real and Complex Matrices By LR and QR Triangularizations," Numerische Mathematik, 16: 181-204 (1970).

Subroutine INPUTA

Purpose:

Reads any real two-dimensional array from either the keyboard or from an existing computer file.

Calling Sequence:

CALL INPUTA (NMAX,N,M,A,DIARY)

Input Variables:

NMAX	The integer number of rows used in the original DIMENSION statement for the matrix A.
N	The integer number of rows of the matrix A. N must not be greater than NMAX.
M	The integer number of columns of the matrix B.
DIARY	The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

A	The real N by M matrix to be entered.

Subroutines Used:

YESORN

Subroutine INVERT

Purpose:

Inverts a real, square matrix.

Calling Sequence:

CALL INVERT (A,AINV,N,DIARY)

Input Variables:

A	The real N by N matrix to be inverted.
N	The integer number of rows of the matrix A. Restriction on N: N≤15.
DIARY	The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

AINV	The real N by N matrix which is the inverse of the matrix A.

If the matrix to be inverted is singular, an error message is printed.

Subroutines Used:

FACTOR and SUBST.

Subroutine PACHDF

Purpose:

Defines the matrix of optimal eigenvectors $\bar{p}_o$ and the product of the feedback gain matrix and the optimal eigenvectors, $G\bar{p}_o$.

Calling Sequence:

CALL PACHDF (PA,PACH,NMAX,N,M,K)

Input Variables:

PA — The real N by 1 array containing the vector $\bar{p}_o$ in the first 2n elements, and the vector $G\bar{p}_o$ in the last 2m elements. Here, n is the number of rows of the square matrix A, and m is the number of columns in the input matrix B for $\dot{\bar{x}} = A\bar{x} + B\bar{u}$.

NMAX — The integer number of rows used in the original DIMENSION statement for the arrays PA and PACH in the calling program. Note: the original DIMENSION statement for both PA and PACH must have the same number of rows.

N — The integer number of rows of the array PACH, and the number of elements in the one-dimensional array PA. Where n and m are defined as in the discussion of PA above, $N = 2(n+m)$. N must not be greater than NMAX.

M — The integer number of rows of the array PACH. Where n and m are defined as in the discussion of

PA above, M = m .

K — The integer identifying which column of PACH is to receive the input data in PA.

Output Variables:

PACH — The real N by M array containing all defined optimal vectors $\bar{p}_o$ in the first 2n rows, and the vectors $G\bar{p}_o$ in the last 2m elements. Here, n and m are defined as in the discussion of PA above.

Subroutine PDDEF

Purpose:

Defines the desired one-dimensional eigenvector array from the two-dimensional array containing all desired eigenvectors.

Calling Sequence:

CALL PDDEF (EVEC,PD,NMAX,N,M,K)

Input Variables:

EVEC	The real N by M array containing all of the desired eigenvectors.
NMAX	The integer number of rows used in the original DIMENSION statement for the arrays EVEC and PD in the calling program. Note: the original DIMENSION statement for both arrays must have the same number of rows.
N	The integer number of rows of the array EVEC, and the number of elements of the aray PA. N must not be greater than NMAX.
M	The integer number of columns of the array EVEC.
K	The integer index defining which column of EVEC will be used to define PD.

Output Variables:

PD	The real N by 1 array containing a desired eigenvector.

Subroutine PLOT

Plots the discrete time response history of a system.

Calling Sequence:

CALL PLOT (XHISTR,KP1,N,T,DIARY)

Input Variables:

XHISTR — The real N by KP1 array containing the time history information of the system. The i^{th} row contains the information for the i^{th} element of the state vector.

KP1 — The integer number of columns of the array XHISTR. Also, the number of discrete points used to describe the time response history of each state.

N — The integer number of rows of the array XHISTR. Also, the number of elements in the state vector used to generate XHISTR. Restriction on N: N≤15.

T — The real time increment used to discretize the system time response history.

DIARY — The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

None.

The time response history is plotted on the terminal screen.

Subroutine Used:

YESORN

Associated Subroutine:

SIMUL8

Subroutine PNRDEF

Purpose:

Defines the matrices P and R to be used to find the required feedback gain matrix.

Calling Statement:

CALL PNRDEF (PACH,P,R,NMAXPA,NMAXP,NMAXR,N,M,EFFECO,DIARY)

Input Variables:

PACH — The real 2(N+M) by N array containing the matrix P in the first 2N rows, and the matrix R in the last 2M rows.

NMAXPA, NMAXP, NMAXR — The integer number of rows used in the original DIMENSION statement for the arrays PACH, P and R, respectively, in the calling program. Restrictions: NMAXPA≤15 ; NMAXP≤15 ; NMAXR≤15 .

N — The integer number of rows and columns in the array P.

M — The integer number of rows in the array R.

EFFECO — The real number used as the tolerance, or the effective value of zero. If the sum of the absolute value of the imaginary components of a vector is less than EFFECO, the vector is assumed to be real.

DIARY — The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be

made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

P	The real N by N array of optimal eigenvectors.
R	The real M by N array used with P to find the required feedback gain matrix.

Subroutine POLZER

Purpose:

Prints the resolvent matrix and the poles and zeroes of the resolvent matrix.

Calling Sequence:

CALL POLZER (A,N,DIARY)

Input Variables:

A	The real N by N matrix, in the time domain, for which the resolvent matrix and poles and zeroes are wanted.
N	The integer number of rows in the square A matrix. Restriction on N: $N \leq 15$.
DIARY	The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

None

The resolvent matrix and poles and zeroes are printed on the terminal.

Subroutines Used:

PROOT, RESOLV, YESOSRN

Subroutine PRINTA

Purpose:

Prints a real matrix.

Calling Sequence:

CALL PRINTA (NMAX,N,M,A,DIARY)

Input Variables:

NMAX	The integer number of rows used in the original DIMENSION statement for the matrix A in the calling program.
N	The integer number of rows in the matrix A.
M	The integer number of columns in the matrix A.
A	The real N by M matrix to be printed.
DIARY	The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

None

Subroutine PRINTZ

Purpose:

Prints the eigenstructure of a matrix as found by subroutines BALANC, BALBAK, EIGV, ELMHES, ELTRAN, and HQR2.

Calling Sequence:

CALL PRINTZ (Z,WR,WI,NMAX,N,DIARY)

Input Variables:

Z	The real N by N matrix of eigenvectors. If the i^{th} eigenvalue is real, the i^{th} column of Z is the associated eigenvector. If the i^{th} eigenvalue is the first of a pair of complex conjugate eigenvalues, the i^{th} column of Z contains the real components of the associated eigenvector and the $i+1^{st}$ column contains the imaginary components.
WR, WI	The real N by 1 arrays containing the real and imaginary components of the eigenvalues, respectively. The i^{th} elements in the arrays correspond to the i^{th} eigenvector in Z. Complex conjugate eigenvalues are stored adjacent to each other.
NMAX	The integer number of rows used in the original DIMENSION statement for the arrays Z, WR, and WI in the calling program. Note: the original DIMENSION statements for each of these arrays must have the same number of rows.
N	The integer number of rows of the Z, WR, and WI arrays. N must not be greater than NMAX.

DIARY The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

None

Subroutines Used:

PRINTA, PRNVEC

Subroutine PRNVEC

Purpose:

Prints a one-dimensional vector.

Calling Sequence:

CALL PRNVEC (S,NMAX,N,DIARY)

Input Variables:

S	The real N by 1 vector to be printed.
NMAX	The integer number of elements used in the original DIMENSION statement for the vector S.
N	The integer number of elements of the vector S.
DIARY	The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

None

Subroutine PROOT

Purpose:

Finds the roots of a polynomial with real coefficients, using a modified Bairstow method.

Calling Sequence:

CALL PROOT (N,A,U,V,IR)

Input Variables:

N — The integer degree of the polynomial.
Restriction: $N \leq 15$.

A — The real N+1 by 1 array of the polynomial coefficients.

IR — The integer indicating how the array A is written.
If IR = +1 , the polynomial is written as

$$A(1) + A(2)x + A(3)x^2 + \ldots + A(N+1)x^N$$

If IR = -1 , the polynomial is written as

$$A(1)x^N + A(2)x^{(N-1)} + \ldots + A(N)x + A(N+1)$$

Output Variables:

U — The real N+1 by 1 array containing the real components of the roots of the polynomial.

V — The real N+1 by 1 array containing the imaginary

components of the roots of the polynomial.

Reference:

Melsa, James L., and Stephen K. Jones. Computer Programs for Computational Assistance in the Study of Linear Control Theory (Second Edition), pp 161-163. New York: McGraw-Hill Book Company, 1973.

Subroutine RESOLV

Purpose:

Calculates the resolvent matrix in the form

$$[sI-A]^{-1} = \frac{Q(s)}{\Delta(s)}$$

Uses the Leverrier or Faddeev recursion.

Calling Sequence:

CALL RESOLV (AR,POLESR,QR,N,CHECK,DIARY)

Input Variables:

AR	The real N by N matrix from which the resolvent matrix is to be defined.
N	The integer number of rows in the square matrix AR. Restriction: N≤15 .
CHECK	The logical variable used to indicate if a check of the accuracy of the subroutine results is to be printed. If CHECK = .TRUE. , the check will be performed and printed. If CHECK = .FALSE. , the check is not performed.
DIARY	The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

POLESR The real N by 1 array containing the coefficients of the characteristic equation. The characteristic equation is written as

$$\Delta(s) = s^N + POLESR(N)s^{N-1} + \ldots + POLESR(2)s + POLESR(1)$$

QR The real N by N by N array containing the coefficients of the N^2 polynomials of the numerator Q(s). The numerator polynomials are written as

$$Q(s) = Q(I,J,N)s^{(N-1)} + Q(I,J,N-1)s^{(N-2)} + \ldots + Q(I,J,2)s + Q(I,J,1)$$

The I,J terms refer to the polynomial in the I,J location of the resolvent matrix numerator.

Subroutine Used:

DAXBEC

Reference:

Reid, J. Gary. Linear System Fundamentals Continuous and Discrete, Classic and Modern, pp 449, 450. New York: McGraw-Hill Book Company, 1983.

Subroutine SAVE

Purpose:

Saves data to a computer file.

Calling Sequence:

CALL SAVE (A,B,C,N,M,DIARY,FILED,ITEST)

Input Variables:

A	The real N by M matrix to be saved.
B	The real N by M matrix to be saved.
C	The real N by 1 matrix to be saved.
N	The integer number of rows of the matrix to be saved. Restriction: If ITEST = 1 N≤15 ; If ITEST > 1 , N≤30 .
M	The integer number of columns of the matrix to be saved.
DIARY	The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.
FILED	The character variable containing the name of the DIARY computer file.
ITEST	The integer indicating which matrix is to be saved. If ITEST = 1 , A is saved. B is saved if ITEST = 2 . If ITEST = 3 , C is saved.

Output Variables:

None

Subroutine SETUPC

Purpose:

Defines the transpose of the augmented matrix $[A-\lambda_i I, B]$, for complex eigenvalues λ_i, to be entered in subroutine SVD.

Calling Sequence:

CALL SETUPC (A,B,EVALR,EVALI,SETUP,N,M,NX2)

Input Variables:

A	The real N by N system matrix used in the augmented matrix.
B	The real N by M input matrix used in the augmented matrix.
EVALR	The real variable representing the real component of the eigenvalue λ_i.
EVALI	The real variable representing the imaginary component of the eigenvalue λ_i.
N	The integer number of rows in the square system matrix A.
M	The integer number of columns in the input matrix B.
NX2	The integer variable equal to 2 times N. NX2 = N x 2 .

Output Variables:

SETUP	The real 2N by 2(N+M) transpose of the augmented matrix.

Subroutine Used:

ATRANS

Subroutine SIGPL

Purpose:

Defines the matrix SIGPLS (Σ^+) from the vector of ordered singular values.

Calling Sequence:

CALL SIGPL (S,SIGPLS,NMAXS,NMAXSP,N,M,EFFECO)

Input Variables:

S	The real N by 1 vector of singular values. The singular values are stored in decreasing order of magnitude.
NMAXS	The integer number of elements used in the original DIMENSION statement for the vector S.
NMAXSP	The integer number of rows used in the original DIMENSION statement for the matrix SIGPLS.
N	The integer number of elements in the vector S, and the number of columns in the matrix SIGPLS. Restrictions: $N \leq NMAXS$.
M	The integer number of rows of the matrix SIGPLS. Restriction: $M \leq NMAXSP$.
EFFECO	The real variables used as the tolerance, or the effective zero.

Output Variable:

SIGPLS	The M by N matrix Σ^+.

Subroutine SIMUL8

Purpose:

Simulates the time response history of a controlled system. Calculates the time response history matrix which is plotted with subroutine PLOT.

Calling Sequence:

CALL SIMUL8 (A,B,N,M,DIARY)

Input Variables:

A	The real N by N system matrix used for the time history simulation.
B	The real N by M input matrix used for the time history simulation.
N	The integer number of rows of the square system matrix A, and the number of rows of the input matrix B. Restrictions: N≤15 .
M	The integer number of columns of the input matrix B. Restrictions: M≤15 .
DIARY	The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

None

Subroutines Used:

AXBEQC, PLOT, YESORN

Subroutine SUBST

Purpose:

Solves for the inverse of a square matrix after the subroutine FACTOR factors the original matrix into a suitable form. Solves for only one column of the inverse matrix per call.

Calling Sequence:

CALL SUBST (W,B,X,IPIVOT,N,NMAX)

Input Variables:

W	The real N by N matrix which has been factored by the subroutine FACTOR. Before being entered to FACTOR, this was the matrix to be inverted.
B	A real N by 1 working array. All elements of B equal 0, except B(J), where J is the number of the column of the inverse matrix to be defined.
IPIVOT	An integer N by 1 working array.
N	The integer number of rows in the square inverse matrix.
NMAX	The integer number of rows used in the original DIMENSION statement for the matrix W in the calling program.

Output Variable:

X	The real N by 1 column of the inverted matrix found. Where B(J) = 1 , X is the J^{th} column of

the inverse matrix.

Associated Subroutines:

FACTOR and INVERT

Subroutine SVD

Purpose:

Calculates the singular value decomposition of a real matrix.

Calling Sequence:

CALL SVD (A,MMAX,NMAX,M,N,NU,NV,S,U,V,IER)

Input Variables:

A	The real M by N matrix for which the singular value decomposition is desired.
MMAX	The integer number of rows used in the original DIMENSION statement for the matrices A and U in the calling program. The original DIMENSION statement for both matrices must have the same number of rows.
M	The integer number of rows of the matrix A.
N	The integer number of columns of the matrix A. Restrictions: $N \leq M$.
NU	The integer number of columns to be calculated for the M by M matrix U. The only values permitted are 0, N, or M.
NV	The integer number of columns to be calculated for the N by N matrix V. The only values permitted are 0 or N.

Output Variables:

S — The real N by 1 vector of singular values. These singular values are sorted in decreasing order of magnitude.

U — The real N by N matrix of the orthonormal eigenvectors of the matrix AA^T.

V — The real M by M matrix of the orthonormal eigenvectors of the matrix A^TA.

IER — The integer indicating when SVD has not converged to a singular value within 30 iterations. If IER=9999 , SVD has not converged. If IER<9999 , SVD has converged.

Reference:

Businger, Peter A., and Gene H. Golub. "Algorithm 358: Singular Value Decomposition of a Complex Matrix," Collected Algorithms From ACM, Vol. 2, Book 1. New York: The Association for Computing Machinery, Inc., 1980.

Subroutine V2DEF

Purpose:

Determines the null space matrix V2 given the matrix V and the ordered singular values. V is found from the singular decomposition of a source matrix A.

Calling Sequence:

CALL V2DEF (V,NMV,N,S,NMS,V2,NV2,EFFEO,DIARY)

Input Variables:

V	The real N by N matrix of the orthornormal eigen-vectors of a matrix A^TA, where the source matrix A has N rows.
NMV	The integer number of rows used in the original DIMENSION statement for the matrices V and V2 in the calling program.
N	The integer number of rows in the matrices V and V2, and the number of elements in the vector S. Restrictions: N≤NMV ; N≤NMS ; N≤NV2 .
S	The N by 1 array containing the ordered vector of singular values. The singular values are stored in descending order of magnitude.
NMS	The integer number of rows used in the original DIMENSION statement for the vector S in the calling program.
EFFECO	The real variable used as the tolerance, or the effective zero.
DIARY	The logical variable indicating if any WRITE

statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variables:

V2	The real N by MV2 matrix defining the null space of the source matrix A.
NV2	The integer number of columns of the matrix V2. Also, the rank of the source matrix A.

Subroutine V2SDEF

Purpose:

Defines the matrix V2S from the first rows of the null space matrix V2. V2S defines the range space of achievable eigenvectors for MODES. The matrix V2 is the null space of a source matrix A.

Calling Sequence:

CALL V2SDEF (V2,V2S,NV2MAX,NV2SMX,N,M)

Input Variables:

V2	The real two-dimensional matrix defining the null space of the source matrix A.
NV2MAX	The integer number of rows used in the original DIMENSION statement for the matrix V2 in the calling program.
NV2SMX	The integer number of rows used in the original DIMENSION statement for the matrix V2S in the calling program.
N	The integer number of rows in the matrix V2S. Restrictions: N≤NV2SMX .
M	The integer number of columns of the matrices V2 and V2S. Also, the rank of the source matrix A.

Output Variables:

V2S	The real N by M matrix defining the range space of achievable eigenvectors for MODES. The elements

of V2S are the same as the elements of the first N rows of V2.

Subroutine YESORN

Purpose:

Finds the yes-or-no answer to the question asked in the calling program.

Calling Sequence:

CALL YESORN (ANSWER,DIARY)

Input Variable:

DIARY — The logical variable indicating if any WRITE statements should be made to a computer file. If DIARY = .TRUE. then the WRITE statement will be made. If DIARY = .FALSE. , the WRITE statement will not be made.

Output Variable:

ANSWER — The logical variable containing the answer to the question asked in the calling program. If ANSWER = .TRUE. , the answer is yes. If ANSWER = .FALSE. , the answer is no.

VI. Conclusions and Recommendations

The control system design program MODES is an accurate, effective tool for the design engineer. By entering the equations of motion for a controlled system in matrix form, and specifying a desired eigenstructure, we can easily find the unique closed-loop feedback gain matrix which alters the system's eigenstructure to be as close to the desired as possible. By presenting the eigenstructure as well as a graphical time response history in the modern control representation of our system, and by presenting the resolvent matrix, poles and zeroes in the classical representation, MODES is not only a good design tool, it is also a good learning aid for students of control system design.

Suggested follow-on efforts include modifying MODES so as to allow the designer to specify only parts of the eigenstructure. Because some mode shapes may be more important to a particular design problem, it may be desired to introduce greater design freedom by leaving some components of some eigenvectors unspecified. This way, the more critical eigenvector components may be placed closer to their desired values. Another follow-on effort would be to incorporate MODES into a comprehensive computer-aided control system design package.

Appendix A: MODES Main Program Listing

The Fortran listing of the main program MODES is presented below. This listing of MODES is the version developed for the VAX/VMS computer. Some modifications, as discussed in the Users' Guide, are required to compile this program for the IBM PC compatible computers.

```
      PROGRAM MODES
      REAL A(15,15),B(15,15),EVEC(30,15),EVAL(30),ATILDE(15,15),
     +P(15,15),R(15,15),PINV(15,15),G(15,15),Z(15,15),WR(15),
     +WI(15),ZTIL(15,15),WRTIL(15),WITIL(15),T,PACH(60,15),
     +SETUP(60,30),PD(30),S(30),U(30,30),V(30,30),V2(60,60),V2S(30,60),
     +SV2S(60),UV2S(30,30),VV2S(60,60),SIGPLS(60,30),TEMP1(60,60),
     +UV2ST(30,30),TEMP2(60,60),PA(60)
      LOGICAL PROMPT,ANSWER,DIARY,OLD
      CHARACTER FILEO*14,TEST*1
      NMAX = 15
      EFFECO = 1.E-7
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      ANSWER = .FALSE.
      WRITE(*,*) 'DO YOU WANT A FILE OF THIS SESSION OF MODES?'
      CALL YESORN(DIARY,ANSWER)
      IF(DIARY) THEN
```

```
      FILEO='JOURN1'
      OPEN(4,FILE=FILEO,STATUS='NEW')
      REWIND 4
      WRITE(4,*) 'DO YOU WANT THE INTRODUCTORY MESSAGE?'
   END IF
   WRITE(*,*) 'DO YOU WANT THE INTRODUCTORY MESSAGE?'
   CALL YESORN(ANSWER,DIARY)
   IF(ANSWER) THEN
    WRITE(*,*) 'THE PROGRAM MODES CALCULATES THE MATRIX OF FEEDBACK'
    WRITE(*,*) 'GAINS REQUIRED TO GIVE A CONTROLLED SYSTEM THE'
    WRITE(*,*) 'RESPONSE CHARACTERISTICS DESIRED BY THE DESIGNER.'
    WRITE(*,*) 'THE DESIRED CHARACTERISTICS ARE SPECIFIED BY'
    WRITE(*,*) 'DEFINING THE DESIRED EIGENSTRUCTURE.'
    WRITE(*,*) ' '
    WRITE(*,*) 'AFTER ENTERING THE ORIGINAL SYSTEM MATRICES, YOU'
    WRITE(*,*) 'HAVE THE OPTION OF SEEING THE TIME RESPONSE'
    WRITE(*,*) 'CHARACTERISTICS'
    WRITE(*,*) 'OF THE ORIGINAL SYSTEM PLOTTED ON THE SCREEN'
    WRITE(*,*) '(A TIME HISTORY SIMULATION), THE EIGENVALUES '
    WRITE(*,*) 'AND EIGENVECTORS (EIGENSTRUCTURE) OF THE '
    WRITE(*,*) 'ORIGINAL SYSTEM PLOTTED ON THE SCREEN, AS WELL AS'
    WRITE(*,*) 'THE RESOLVENT MATRIX AND POLES AND ZEROS.'
    WRITE(*,*) 'THEN YOU ARE PROMPTED FOR THE DESIRED'
    WRITE(*,*) 'EIGENSTRUCTURE.'
    WRITE(*,*) ' '
    WRITE(*,*) 'HIT THE RETURN KEY WHEN READY TO CONTINUE.'
    READ(*,'(F5.2)') T
    WRITE(*,*) ' '
    WRITE(*,*) ' '
    WRITE(*,*) ' '
    WRITE(*,*) 'AFTER CALCULATING THE REQUIRED FEEDBACK GAIN MATRIX,'
    WRITE(*,*) 'MODES CALCULATES THE ACTUAL EIGENSTRUCTURE OF THE'
    WRITE(*,*) 'MODIFIED SYSTEM, AND OFFERS TO PRINT BOTH THE'
    WRITE(*,*) 'DESIRED AND THE FINAL EIGENSTRUCTURE.'
    WRITE(*,*) ' '
    WRITE(*,*) 'YOU ARE THEN ASKED IF YOU WANT THE TIME RESPONSE'
    WRITE(*,*) 'CHARACTERISTICS AND THE RESOLVENT MATRIX AND POLES'
    WRITE(*,*) 'AND ZEROS PLOTTED ON THE SCREEN.  THROUGHOUT MODES,'
    WRITE(*,*) 'YOU ARE ASKED IF YOU WANT TO SAVE THE DATA YOU HAVE'
    WRITE(*,*) 'ENTERED.'
    WRITE(*,*) ' '
    WRITE(*,*) 'YOU ARE LIMITED TO 15 STATES AND 15 INPUTS.'
    WRITE(*,*) ' '
    WRITE(*,*) 'HIT THE RETURN KEY WHEN READY TO CONTINUE.'
   IF(DIARY) THEN
    WRITE(4,*) 'THE PROGRAM MODES CALCULATES THE MATRIX OF FEEDBACK'
    WRITE(4,*) 'GAINS REQUIRED TO GIVE A CONTROLLED SYSTEM THE'
    WRITE(4,*) 'RESPONSE CHARACTERISTICS DESIRED BY THE DESIGNER.'
    WRITE(4,*) 'THE DESIRED CHARACTERISTICS ARE SPECIFIED BY'
    WRITE(4,*) 'DEFINING THE DESIRED EIGENSTRUCTURE.'
    WRITE(4,*) ' '
    WRITE(4,*) 'AFTER ENTERING THE ORIGINAL SYSTEM MATRICES, YOU'
    WRITE(4,*) 'HAVE THE OPTION OF SEEING THE TIME RESPONSE'
    WRITE(4,*) 'CHARACTERISTICS'
```

```
      WRITE(4,*) 'OF THE ORIGINAL SYSTEM PLOTTED ON THE SCREEN'
      WRITE(4,*) '(A TIME HISTORY SIMULATION), THE EIGENVALUES '
      WRITE(4,*) 'AND EIGENVECTORS (EIGENSTRUCTURE) OF THE '
      WRITE(4,*) 'ORIGINAL SYSTEM PLOTTED ON THE SCREEN, AS WELL AS'
      WRITE(4,*) 'THE RESOLVENT MATRIX AND POLES AND ZEROS.'
      WRITE(4,*) 'THEN YOU ARE PROMPTED FOR THE DESIRED'
      WRITE(4,*) 'EIGENSTRUCTURE.'
      WRITE(4,*) ' '
      WRITE(4,*) 'HIT THE RETURN KEY WHEN READY TO CONTINUE.'
      WRITE(4,*) ' '
      WRITE(4,*) ' '
      WRITE(4,*) ' '
      WRITE(4,*) 'AFTER CALCULATING THE REQUIRED FEEDBACK GAIN MATRIX,'
      WRITE(4,*) 'MODES CALCULATES THE ACTUAL EIGENSTRUCTURE OF THE'
      WRITE(4,*) 'MODIFIED SYSTEM, AND OFFERS TO PRINT BOTH THE'
      WRITE(4,*) 'DESIRED AND THE FINAL EIGENSTRUCTURE.'
      WRITE(4,*) ' '
      WRITE(4,*) 'YOU ARE THEN ASKED IF YOU WANT THE TIME RESPONSE'
      WRITE(4,*) 'CHARACTERISTICS AND THE RESOLVENT MATRIX AND POLES'
      WRITE(4,*) 'AND ZEROS PLOTTED ON THE SCREEN.  THROUGHOUT MODES,'
      WRITE(4,*) 'YOU ARE ASKED IF YOU WANT TO SAVE THE DATA YOU HAVE'
      WRITE(4,*) 'ENTERED.'
      WRITE(4,*) ' '
      WRITE(4,*) 'YOU ARE LIMITED TO 15 STATES AND 15 INPUTS.'
      WRITE(4,*) ' '
      WRITE(4,*) 'HIT THE RETURN KEY WHEN READY TO CONTINUE.'
     END IF
      READ(*,'(F5.2)') T
     END IF
     WRITE(*,*) ' '
     WRITE(*,*) ' '
     WRITE(*,*) ' '
     WRITE(*,*) 'DO YOU WANT ALL INTERMEDIATE RESULTS?'
     IF(DIARY) THEN
     WRITE(4,*) ' '
     WRITE(4,*) ' '
     WRITE(4,*) ' '
     WRITE(4,*) 'DO YOU WANT ALL INTERMEDIATE RESULTS?'
     END IF
     CALL YESORN(PROMPT,DIARY)
30   WRITE(*,*) ' '
     IF(DIARY) WRITE(4,*) ' '
     CALL DIMENA(N,M,DIARY)
     IF(N .GT. 15) THEN
        WRITE(*,*) 'A CANNOT BE LARGER THAN 15 BY 15'
        IF(DIARY) WRITE(4,*) 'A CANNOT BE LARGER THAN 15 BY 15'
        GO TO 30
     END IF
     IF (M .GT. N) THEN
        WRITE(*,*) 'B CANNOT HAVE MORE COLUMNS THAN A'
        IF(DIARY) WRITE(4,*) 'B CANNOT HAVE MORE COLUMNS THAN A'
        GO TO 30
     END IF
     M1 = N + M
```

```
WRITE(*,*) 'ENTER THE ELEMENTS OF MATRIX A'
IF(DIARY) WRITE(4,*) 'ENTER THE ELEMENTS OF MATRIX A'
CALL INPUTA(NMAX,N,N,A,DIARY)
WRITE(*,*) 'DO YOU WANT TO SAVE THIS A MATRIX?'
IF(DIARY) WRITE(4,*) 'DO YOU WANT TO SAVE THIS A MATRIX?'
CALL YESORN(ANSWER,DIARY)
IF(ANSWER) THEN
   FILEO='JOURN2'
   CALL SAVE(A,EVEC,EVAL,N,N,DIARY,FILEO,1)
END IF
WRITE(*,*) 'ENTER THE ELEMENTS OF MATRIX B'
IF(DIARY) WRITE(4,*) 'ENTER THE ELEMENTS OF MATRIX B'
CALL INPUTA(NMAX,N,M,B,DIARY)
WRITE(*,*) 'DO YOU WANT TO SAVE THIS B MATRIX?'
IF(DIARY) WRITE(4,*) 'DO YOU WANT TO SAVE THIS B MATRIX?'
CALL YESORN(ANSWER,DIARY)
IF(ANSWER) THEN
   FILEO='JOURN3'
   CALL SAVE(B,EVEC,EVAL,N,M,DIARY,FILEO,1)
END IF
WRITE(*,*) 'DO YOU WANT TO SEE THE EIGENSTRUCTURE OF THE'
WRITE(*,*) 'MATRIX A OF THIS SYSTEM?'
IF(DIARY) THEN
   WRITE(4,*) 'DO YOU WANT TO SEE THE EIGENSTRUCTURE OF THE'
   WRITE(4,*) 'MATRIX A OF THIS SYSTEM?'
END IF
CALL YESORN(ANSWER,DIARY)
IF (ANSWER) THEN
  CALL EIGV(A,Z,WR,WI,NMAX,N,DIARY)
  CALL PRINTZ(Z,WR,WI,NMAX,N,DIARY)
END IF
WRITE(*,*) 'DO YOU WANT TO HAVE THE TIME RESPONSE OF THIS'
WRITE(*,*) 'SYSTEM PLOTTED ON THE SCREEN?'
IF(DIARY) THEN
WRITE(4,*) 'DO YOU WANT TO HAVE THE TIME RESPONSE OF THIS'
WRITE(4,*) 'SYSTEM PLOTTED ON THE SCREEN?'
END IF
CALL YESORN(ANSWER,DIARY)
IF (ANSWER) CALL SIMUL8(A,B,N,M,NMAX,DIARY)
WRITE(*,*) 'DO YOU WANT THE RESOLVENT MATRIX FOR THIS SYSTEM?'
IF(DIARY) THEN
   WRITE(4,*) 'DO YOU WANT THE RESOLVENT MATRIX FOR THIS SYSTEM?'
END IF
CALL YESORN(ANSWER,DIARY)
IF (ANSWER) CALL POLZER(A,B,N,M,DIARY)
WRITE(*,*) 'NOW ENTER THE DESIRED EIGENSTRUCTURE.'
IF(DIARY) WRITE(4,*) 'NOW ENTER THE DESIRED EIGENSTRUCTURE.'
CALL EIGSTR(EVEC,EVAL,2*NMAX,N,DIARY)
WRITE(*,*) 'DO YOU WANT TO SAVE THIS DESIRED EIGENSTRUCTURE?'
IF(DIARY) THEN
   WRITE(4,*) 'DO YOU WANT TO SAVE THIS DESIRED EIGENSTRUCTURE?'
END IF
CALL YESORN(ANSWER,DIARY)
IF(ANSWER) THEN
```

```
            WRITE(*,*) 'FOR THE EIGENVECTOR MATRIX,'
            IF(DIARY) WRITE(4,*) 'FOR THE EIGENVECTOR MATRIX,'
            FILEO='JOURN4'
            CALL SAVE(A,EVEC,EVAL,2*N,N,DIARY,FILEO,2)
            WRITE(*,*) 'FOR THE EIGENVALUE MATRIX,'
            IF(DIARY) WRITE(4,*) 'FOR THE EIGENVALUE MATRIX,'
            FILEO='JOURN5'
            CALL SAVE(A,EVEC,EVAL,2*N,1,DIARY,FILEO,3)
         END IF
         IF (PROMPT) THEN
            WRITE(*,*) 'NOW ENTERING THE DO-LOOP'
            IF(DIARY) THEN
               WRITE(4,*) 'NOW ENTERING THE DO-LOOP'
               WRITE(4,*) ' '
            END IF
         END IF
         WRITE(*,*) ' '
         DO 40 I = 1, N
            EVALR = EVAL(I)
            EVALI = EVAL(I+N)
            IF (PROMPT) THEN
               WRITE(*,36) I
               WRITE(*,37) EVALR
               WRITE(*,38) EVALI
               IF(DIARY) THEN
                  WRITE(4,36) I
                  WRITE(4,37) EVALR
                  WRITE(4,38) EVALI
               END IF
36             FORMAT(1X,'I = ',I2)
37             FORMAT(1X,'THE REAL PART OF THE EIGENVALUE IS ',F15.8)
38             FORMAT(1X,'THE IMAGINARY PART IS ',F15.8)
            END IF
            CALL PDDEF(EVEC,PD,2*NMAX,2*N,N,I)
            CALL SETUPC(A,B,EVALR,EVALI,SETUP,N,M,2*N)
            CALL SVD(SETUP,4*NMAX,2*NMAX,2*M1,2*N,2*M1,2*N,S,V,U,IER)
            CALL V2DEF(V,4*NMAX,2*M1,S,2*NMAX,V2,MV2,EFFECO,DIARY)
            CALL V2SDEF(V2,V2S,4*NMAX,2*NMAX,2*N,MV2)
            CALL SVD(V2S,2*NMAX,4*NMAX,2*N,MV2,2*N,MV2,SV2S,UV2S,VV2S,IER)
            CALL ATRANS(2*NMAX,2*NMAX,2*N,2*N,UV2S,UV2ST)
            CALL SIGPL(SV2S,SIGPLS,4*NMAX,4*NMAX,2*N,MV2,EFFECO)
            CALL AXBEQC(VV2S,4*NMAX,MV2,MV2,SIGPLS,4*NMAX,MV2,2*N,TEMP1,
           +4*NMAX,DIARY)
            CALL AXBEQC(TEMP1,4*NMAX,MV2,2*N,UV2ST,2*NMAX,2*N,2*N,TEMP2,
           +4*NMAX,DIARY)
            CALL AXBEQC(V2,4*NMAX,2*M1,MV2,TEMP2,4*NMAX,MV2,2*N,TEMP1,
           +4*NMAX,DIARY)
            CALL AXBEQC(TEMP1,4*NMAX,2*M1,2*N,PD,2*NMAX,2*N,1,PA,4*NMAX,
           +DIARY)
            CALL PACHDF(PA,PACH,4*NMAX,2*M1,N,I)
            IF (PROMPT) THEN
               WRITE(*,*) ' '
               WRITE(*,*) 'THE MATRIX PACH IS'
               IF(DIARY) THEN
```

```
               WRITE(4,*) ' '
               WRITE(4,*) 'THE MATRIX PACH IS'
            END IF
            CALL PRINTA(4*NMAX,2*M1,N,PACH,DIARY)
         END IF
40    CONTINUE
      IF (PROMPT) THEN
         WRITE(*,*) 'END OF DO-LOOP'
         IF(DIARY) WRITE(4,*) 'END OF DO-LOOP'
      END IF
      CALL PNRDEF(PACH,P,R,60,15,15,N,M,EFFECO,DIARY)
      IF (PROMPT) THEN
        WRITE(*,*) 'THE MATRIX P IS'
        IF(DIARY) WRITE(4,*) 'THE MATRIX P IS'
        CALL PRINTA(15,N,N,P,DIARY)
        WRITE(*,*) 'THE MATRIX R IS'
        IF(DIARY) WRITE(4,*) 'THE MATRIX R IS'
        CALL PRINTA(15,M,N,R,DIARY)
      END IF
      CALL INVERT(P,PINV,N,DIARY)
      IF (PROMPT) THEN
        WRITE(*,*) 'THE INVERSE OF P IS PINV:'
        IF(DIARY) WRITE(4,*) 'THE INVERSE OF P IS PINV:'
        CALL PRINTA(15,N,N,PINV,DIARY)
      END IF
      CALL AXBEQC(R,15,M,N,PINV,15,N,N,G,15,DIARY)
      WRITE(*,*) 'THE CALCULATED FEEDBACK GAIN MATRIX G IS'
      IF(DIARY) WRITE(4,*) 'THE CALCULATED FEEDBACK GAIN MATRIX G IS'
      CALL PRINTA(NMAX,M,N,G,DIARY)
      WRITE(*,*) 'DO YOU WANT TO SAVE THIS GAIN MATRIX?'
      IF(DIARY) WRITE(4,*) 'DO YOU WANT TO SAVE THIS GAIN MATRIX?'
      CALL YESORN(ANSWER,DIARY)
      IF(ANSWER) THEN
        FILEO='JOURN6'
        CALL SAVE(G,EVEC,EVAL,M,N,DIARY,FILEO,1)
      END IF
      CALL ATLDEF(A,B,G,ATILDE,NMAX,N,M,DIARY)
      WRITE(*,*) 'THE NEW MATRIX A-TILDE IS'
      IF(DIARY) WRITE(4,*) 'THE NEW MATRIX A-TILDE IS'
      CALL PRINTA(NMAX,N,N,ATILDE,DIARY)
      WRITE(*,*) 'DO YOU WANT TO SAVE THIS A-TILDE MATRIX?'
      IF(DIARY)WRITE(4,*) 'DO YOU WANT TO SAVE THIS A-TILDE MATRIX?'
      CALL YESORN(ANSWER,DIARY)
      IF(ANSWER) THEN
        FILEO='JOURN7'
        CALL SAVE(ATILDE,EVEC,EVAL,N,N,DIARY,FILEO,1)
      END IF
      WRITE(*,*) 'DO YOU WANT TO SEE THE EIGENSTRUCTURE OF THE'
      WRITE(*,*) 'NEW, MODIFIED SYSTEM?'
      IF(DIARY) THEN
         WRITE(4,*) 'DO YOU WANT TO SEE THE EIGENSTRUCTURE OF THE'
         WRITE(4,*) 'NEW, MODIFIED SYSTEM?'
      END IF
      CALL YESORN(ANSWER,DIARY)
```

```
IF (ANSWER) THEN
WRITE(*,*) 'THE EIGENSTRUCTURE OF A-TILDE = A-BG IS'
IF(DIARY) THEN
   WRITE(4,*) 'THE EIGENSTRUCTURE OF A-TILDE = A-BG IS'
END IF
CALL EIGV(ATILDE,ZTIL,WRTIL,WITIL,NMAX,N,DIARY)
CALL PRINTZ(ZTIL,WRTIL,WITIL,NMAX,N,DIARY)
END IF
WRITE(*,*) 'DO YOU WANT TO SEE THE EIGENSTRUCTURE OF THE'
WRITE(*,*) 'ORIGINAL, UNMODIFIED SYSTEM?'
IF(DIARY) THEN
   WRITE(4,*) 'DO YOU WANT TO SEE THE EIGENSTRUCTURE OF THE'
   WRITE(4,*) 'ORIGINAL, UNMODIFIED SYSTEM?'
END IF
CALL YESORN(ANSWER,DIARY)
IF (ANSWER) THEN
  CALL EIGV(A,Z,WR,WI,NMAX,N,DIARY)
  CALL PRINTZ(Z,WR,WI,NMAX,N,DIARY)
END IF
WRITE(*,*) 'DO YOU WANT THE TIME RESPONSE OF THIS MODIFIED'
WRITE(*,*) 'SYSTEM PLOTTED ON THE SCREEN?'
IF(DIARY) THEN
   WRITE(4,*) 'DO YOU WANT THE TIME RESPONSE OF THIS MODIFIED'
   WRITE(4,*) 'SYSTEM PLOTTED ON THE SCREEN?'
END IF
CALL YESORN(ANSWER,DIARY)
IF (ANSWER) CALL SIMUL8(ATILDE,B,N,M,NMAX,DIARY)
WRITE(*,*) 'DO YOU WANT THE RESOLVENT MATRIX FOR THE MODIFIED'
WRITE(*,*) 'SYSTEM?'
IF(DIARY) THEN
   WRITE(4,*) 'DO YOU WANT THE RESOLVENT MATRIX FOR THE MODIFIED'
   WRITE(4,*) 'SYSTEM?'
END IF
CALL YESORN(ANSWER,DIARY)
IF (ANSWER) CALL POLZER(ATILDE,B,N,M,DIARY)
IF(DIARY) CLOSE(4)
END
```

Appendix B: MODES Subroutines Listing

The Fortran listing of the subroutines for MODES is presented below. This listing is the version developed for the VAX/VMS computer. Some modifications, as discussed in the Users' Guide, are required to compile this program for the IBM PC compatible computers.

```
C*********************************************************************|
      SUBROUTINE ATLDEF(A,B,G,ATILDE,NM,N,M,DIARY)
C                                                                     |
C     DEFINES THE MATRIX A-TILDE FROM THE MATRICES A, B, AND G        |
C*********************************************************************|
      REAL A(NM,N),B(NM,M),G(NM,N),ATILDE(NM,N)
      LOGICAL DIARY
      CALL AXBEQC(B,NM,N,M,G,NM,M,N,ATILDE,NM,DIARY)
      DO 10 I = 1, N
         DO 10 J = 1, N
            ATILDE(I,J) = A(I,J) + ATILDE(I,J)
  10  CONTINUE
       WRITE(*,*) ' '
       WRITE(*,*) ' '
      RETURN
      END
C*********************************************************************|
      SUBROUTINE ATRANS(NM,MN,N,M,A,AT)
C                                                                     |
C     DEFINING A-TRANSPOSE, AT(M,N)                                   |
C*********************************************************************|
      REAL A(NM,M),AT(MN,N)
      DO 20 I=1,N
          DO 10 J=1,M
              AT(J,I)=A(I,J)
 10       CONTINUE
 20   CONTINUE
      RETURN
      END
C*********************************************************************|
      SUBROUTINE AXBEQC(A,NMAXA,NA,MA,B,NMAXB,NB,MB,C,NMAXC,DIARY)
C                                                                     |
C     MULTIPLIES MATRICES A AND B TO YIELD MATRIX C                   |
C*********************************************************************|
      REAL B(NMAXB,MB),A(NMAXA,MA),C(NMAXC,MB)
      LOGICAL DIARY
```

```
      IF (MA .EQ. NB) THEN
          DO 10 K = 1, NA
               DO 10 J = 1, MB
                    C(K,J) = 0.0
                    DO 10 I = 1, MA
                         C(K,J) = C(K,J) + A(K,I) * B(I,J)
 10       CONTINUE
      ELSE
          WRITE(*,*) 'THE MATRIX DIMENSIONS ARE NOT COMPATIBLE FOR'
          WRITE(*,*) 'MULTIPLICATION'
          WRITE(*,*) 'THE FIRST MATRIX IS',NA,'     BY',MA
          WRITE(*,*) 'THE SECOND MATRIX IS',NB,'     BY',MB
          IF(DIARY) THEN
             WRITE(4,*) 'THE MATRIX DIMENSIONS ARE NOT COMPATIBLE FOR'
             WRITE(4,*) 'MULTIPLICATION'
             WRITE(4,*) 'THE FIRST MATRIX IS',NA,'     BY',MA
             WRITE(4,*) 'THE SECOND MATRIX IS',NB,'     BY',MB
          END IF
      END IF
      RETURN
      END
C*********************************************************************|
      SUBROUTINE BALANC(NM,N,A,LOW,IGH,SCALE)
C                                                                     |
C     BALANCES A REAL GENERAL MATRIX TO BE USED WITH HQR2, ELTRAN,    |
C     ELMHES, AND BALBAK TO FIND THE EIGENVALUES AND EIGENVECTORS     |
C*********************************************************************|
      REAL A(NM,N),SCALE(N)
      REAL C,F,G,R,S,B2,RADIX
      REAL ABS
      LOGICAL NOCONV
C
C     ********** RADIX IS A MACHINE DEPENDENT PARAMETER SPECIFYING
C                THE BASE OF THE MACHINE FLOATING POINT REPRESENTATION.
C
C                **********
      RADIX = 16.
C
      B2 = RADIX * RADIX
      K = 1
      L = N
      GO TO 100
C     **********IN-LINE PROCEDURE FOR ROW AND
C                COLUMN EXCHANGE **********
  20  SCALE(M) = J
      IF (J .EQ. M) GO TO 50
C
      DO 30 I = 1, L
         F = A(I,J)
         A(I,J) = A(I,M)
         A(I,M) = F
  30  CONTINUE
C
      DO 40 I = K, N
```

```
            F = A(J,I)
            A(J,I) = A(M,I)
            A(M,I) = F
   40    CONTINUE
C
   50    GO TO (80,130), IEXC
C        ********** SEARACH FOR ROWS ISOLATING AN EIGENVALUE
C                AND PUSH THEM DOWN **********
   80    IF (L .EQ. 1) GO TO 280
         L = L - 1
C        ********** FOR J=L STEP -1 UNTIL 1 DO -- **********
  100    DO 120 JJ = 1, L
            J = L + 1 - JJ
C
            DO 110 I = 1, L
               IF (I .EQ. J) GO TO 110
               IF (A(J,I) .NE. 0.0) GO TO 120
  110       CONTINUE
C
            M = L
            IEXC = 1
            GO TO 20
  120    CONTINUE
C
         GO TO 140
C        ********** SEARCH FOR COLUMNS ISOLATING AN EIGENVALUE
C                  AND PUSH THEM LEFT **********
C
  130    K = K + 1
C
  140    DO 170 J = K, L
C
            DO 150 I = K, L
               IF (I .EQ. J) GO TO 150
               IF (A(I,J) .NE. 0.0) GO TO 170
  150       CONTINUE
C
            M = K
            IEXC = 2
            GO TO 20
  170    CONTINUE
C        ********** NOW BALANCE THE SUBMATRIX IN ROWS K TO L **********
         DO 180 I = K, L
  180    SCALE(I) = 1.0
C        ********** ITERATIVE LOOP FOR NORM REDUCTION **********
  190    NOCONV = .FALSE.
C
         DO 270 I = K, L
            C = 0.0
            R = 0.0
C
            DO 200 J = K, L
               IF (J .EQ. I) GO TO 200
               C = C + ABS(A(J,I))
```

```
            R = R + ABS(A(I,J))
 200     CONTINUE
C    ********** GUARD AGAINST ZERO C OR R DUE TO UNDERFLOW **********
         IF (C .EQ. 0.0 .OR. R .EQ. 0.0) GO TO 270
         G = R / RADIX
         F = 1.0
         S = C + R
 210     IF (C .GE. G) GO TO 220
         F = F * RADIX
         C = C * B2
         GO TO 210
 220     G = R * RADIX
 230     IF (C .LT. G) GO TO 240
         F = F / RADIX
         C = C / B2
         GO TO 230
C   ********** NOW BALANCE **********
 240     IF ((C + R) / F .GE. 0.95 * S ) GO TO 270
         G = 1.0 / F
         SCALE(I) = SCALE(I) * F
         NOCONV = .TRUE.
C
         DO 250 J = K, N
 250     A(I,J) = A(I,J) * G
C
         DO 260 J = 1, L
 260     A(J,I) = A(J,I) * F
C
 270  CONTINUE
C
      IF (NOCONV) GO TO 190
C
 280  LOW = K
      IGH = L
      RETURN
      END
C*********************************************************************|
      SUBROUTINE BALBAK(NM,N,LOW,IGH,SCALE,M,Z)
C                                                                     |
C     TO BACK TRANSFORM THE EIGENVECTORS OF THE REAL GENERAL          |
C               MATRIX TRANSFORMED BY BALANC                          |
C*********************************************************************|
      INTEGER I,J,K,M,N,II,NM,IGH,LOW
      REAL SCALE(N),Z(NM,M)
      REAL S
C
      IF (M .EQ. 0) GO TO 200
      IF (IGH .EQ. LOW) GO TO 120
C
      DO 110 I = LOW, IGH
         S = SCALE(I)
C     ********** LEFT HAND EIGENVECTORS ARE BACK TRANSFORMED
C                IF THE FOREGOING STATEMENT IS REPLACED BY
C                S=1.0/SCALE(I). **********
```

```
         DO 100 J = 1, M
 100     Z(I,J) = Z(I,J) * S
C
 110  CONTINUE
C
C     ********- FOR I=LOW-1 STEP -1 UNTIL 1,
C                IGH+1 STEP 1 UNTIL N DO -- **********
 120  DO 140 II = 1, N
         I = II
         IF (I .GE. LOW .AND. I .LE. IGH) GO TO 140
         IF (I .LT. LOW) I = LOW - II
         K = SCALE(I)
         IF (K .EQ. I) GO TO 140
C
         DO 130 J = 1, M
            S = Z(I,J)
            Z(I,J) = Z(K,J)
            Z(K,J) = S
 130     CONTINUE
C
 140  CONTINUE
C
 200  RETURN
      END
C*********************************************************************|
      SUBROUTINE COPYAB(NMAX,N,M,A,B)
C                                                                     |
C     DEFINES MATRIX B = MATRIX A                                     |
C*********************************************************************|
      REAL A(NMAX,M),B(NMAX,M)
      DO 20 I = 1, N
         DO 10 J = 1, M
            B(I,J) = A(I,J)
  10     CONTINUE
  20  CONTINUE
      RETURN
      END
C*********************************************************************|
      SUBROUTINE DAXBEC(A,NMAXA,NA,MA,B,NMAXB,NB,MB,C,NMAXC,DIARY)
C                                                                     |
C     A DOUBLE PRECISION ROUTINE TO MULTIPLY TWO MATRICES (A&B)       |
C*********************************************************************|
      DOUBLE PRECISION A(NMAXA,MA),B(NMAXB,MB),C(NMAXC,MB)
      LOGICAL DIARY
      IF(MA .EQ. NB) THEN
         DO 10 K = 1, NA
            DO 10 J = 1, MB
               C(K,J) = 0.DO
               DO 10 I = 1, MA
                  C(K,J) = C(K,J) + A(K,I) * B(I,J)
  10     CONTINUE
      ELSE
         WRITE(*,*)'THE MATRIX DIMENSIONS ARE NOT COMPATIBLE FOR'
         WRITE(*,*)'MULTIPLICATION'
```

```
         WRITE(*,*)'THE FIRST MATRIX IS',NA,'  BY',MA
         WRITE(*,*)'THE SECOND MATRIX IS',NB,'  BY',MB
         IF(DIARY) THEN
            WRITE(4,*)'THE MATRIX DIMENSIONS ARE NOT COMPATIBLE FOR'
            WRITE(4,*)'MULTIPLICATION'
            WRITE(4,*)'THE FIRST MATRIX IS',NA,'  BY',MA
            WRITE(4,*)'THE SECOND MATRIX IS',NB,'  BY',MB
         END IF
      END IF
      RETURN
      END
C*********************************************************************|
      SUBROUTINE DIMENA(N,M,DIARY)
C                                                                     |
C     THIS SUBROUTINE READS IN N AND M                                |
C*********************************************************************|
      INTEGER M,N
      LOGICAL ANSWER,DIARY
 10   WRITE(*,*) 'WHAT IS THE NUMBER OF ROWS AND COLUMNS (N) OF'
      WRITE(*,*) 'THE SQUARE MATRIX A (THE NUMBER OF STATES)?'
      READ (*,20) N
 20   FORMAT(I2)
      IF(DIARY) THEN
         WRITE(4,*) 'WHAT IS THE NUMBER OF ROWS AND COLUMNS (N) OF'
         WRITE(4,*) 'THE SQUARE MATRIX A (THE NUMBER OF STATES)?'
         WRITE(4,*) N
         WRITE(4,*) 'HOW MANY COLUMNS (M) OF THE N BY M MATRIX B'
         WRITE(4,*) '(THE NUMBER OF INPUTS)?'
      END IF
      WRITE(*,*) 'HOW MANY COLUMNS (M) OF THE N BY M MATRIX B'
      WRITE(*,*) '(THE NUMBER OF INPUTS)?'
      READ(*,30) M
 30   FORMAT(I2)
      IF(DIARY) THEN
         WRITE(4,*) M
         WRITE(4,*) 'THE SYSTEM HAS ',N,' STATES AND ',M,' INPUTS'
         WRITE(4,*) 'IS THIS CORRECT?'
      END IF
      WRITE(*,*) 'THE SYSTEM HAS ',N,' STATES AND ',M,' INPUTS'
      WRITE(*,*) 'IS THIS CORRECT?'
      CALL YESORN(ANSWER,DIARY)
      IF(ANSWER) THEN
         RETURN
      ELSE
         GO TO 10
      END IF
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
      END IF
      RETURN
      END
```

```
C*********************************************************************|
      SUBROUTINE EIGSTR(EVEC,EVAL,NMX2,N,DIARY)
C                                                                     |
C     DEFINES THE DESIRED EIGENSTRUCTURE AND DEFINES THE NEEDED       |
C     MATRICES FROM THIS DEFINITION                                   |
C*********************************************************************|
      REAL EVEC(NMX2,N),EVAL(NMX2)
      LOGICAL COMP,ANSWER,DIARY,KBRD
      CHARACTER TEST*1,FILEI*14,FILEII*14
      COMP=.FALSE.
      N2 = N+N
      WRITE(*,*) ' '
      WRITE(*,*) ' '
  1   WRITE(*,*) 'DO YOU WANT TO USE A NEW EIGENSTRUCTURE MATRIX'
      WRITE(*,*) 'ENTERED FROM THE KEYBOARD, '
      WRITE(*,*) 'OR ONE ALREADY STORED ON THE COMPUTER?'
      WRITE(*,*) ' '
      WRITE(*,*) 'ENTER:    K FOR KEYBOARD    OR    C FOR COMPUTER'
      IF(DIARY) THEN
      WRITE(4,*) ' '
      WRITE(4,*) ' '
      WRITE(4,*) 'DO YOU WANT TO ENTER THE EIGENSTRUCTURE FROM'
      WRITE(4,*) 'THE KEYBOARD, OR FROM MATRICES ALREADY'
      WRITE(4,*) '      STORED ON THE COMPUTER?'
      WRITE(4,*) ' '
      WRITE(4,*) 'ENTER:    K FOR KEYBOARD    OR    C FOR COMPUTER'
      END IF
      READ(*,'(A1)') TEST
      IF(DIARY) WRITE(4,*) TEST
      KBRD = ((LLE(TEST,'K') .AND. LGE(TEST,'K')) .OR.
     +(LLE(TEST,'k') .AND. LGE(TEST,'k')))
      IF (KBRD) GO TO 2
      IF((LLT(TEST,'C') .OR. LGT(TEST,'C')) .AND.
     +(LLT(TEST,'c') .OR. LGT(TEST,'c'))) GO TO 1
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      IF(DIARY) THEN
      WRITE(4,*) ' '
      WRITE(4,*) ' '
      WRITE(4,*) ' '
      END IF
      GO TO 500
C
C   OPTION 1:  ENTERING DATA FROM THE KEYBOARD
C
   2  WRITE(*,*) ' '
      WRITE(*,*) 'COMPLEX EIGENVECTORS AND EIGENVALUES MUST APPEAR IN'
      WRITE(*,*) 'COMPLEX CONJUGATE PAIRS.  WHEN YOU ENTER ONE COMPLEX'
      WRITE(*,*) 'VECTOR OR VALUE, THE COMPLEX CONJUGATE WILL BE'
      WRITE(*,*) 'DEFINED BY THE PROGRAM AUTOMATICALLY.'
      WRITE(*,*) ' '
      IF(DIARY) THEN
      WRITE(4,*) ' '
```

```
      WRITE(4,*) 'COMPLEX EIGENVECTORS AND EIGENVALUES MUST APPEAR IN'
      WRITE(4,*) 'COMPLEX CONJUGATE PAIRS.  WHEN YOU ENTER ONE COMPLEX'
      WRITE(4,*) 'VECTOR OR VALUE, THE COMPLEX CONJUGATE WILL BE'
      WRITE(4,*) 'DEFINED BY THE PROGRAM AUTOMATICALLY.'
      WRITE(4,*) ' '
      END IF
      TTEST=1.0
      DO 400 K=1,N
          IF(TTEST .LT. 0.0) GOTO 300
10        WRITE(*,15) K
          IF(DIARY) WRITE(4,15) K
15        FORMAT(1X,'IS THE DESIRED EIGENVALUE NUMBER ',I2,' COMPLEX?')
          CALL YESORN(ANSWER,DIARY)
          IF(ANSWER) THEN
           IF(K.EQ.N) THEN
                WRITE(*,*) 'YOU ALREADY HAVE TOO MANY EIGENVALUES TO'
                WRITE(*,*) 'DEFINE A COMPLEX CONJUGATE.'
       IF(DIARY) WRITE(4,*) 'YOU ALREADY HAVE TOO MANY EIGENVALUES TO'
       IF(DIARY) WRITE(4,*) 'DEFINE A COMPLEX CONJUGATE.'
           GO TO 10
           END IF
           COMP = .TRUE.
           WRITE(*,16)
           WRITE(*,17) K
           IF(DIARY) WRITE(4,16)
           IF(DIARY) WRITE(4,17) K
16         FORMAT(1X,'THE REAL PART OF THE DESIRED')
17         FORMAT(1X,'EIGENVALUE NUMBER',I2,' = ?')
           READ(*,*) EVAL(K)
           EVAL(K+1) = EVAL(K)
           IF(DIARY) THEN
              WRITE(4,21) EVAL(K)
21            FORMAT(1X,F15.7)
              WRITE(4,*) 'THE IMAGINARY PART = ?'
           END IF
           WRITE(*,*) 'THE IMAGINARY PART = ?'
           READ(*,*) EVAL(K+N)
           EVAL(K+N+1) = -EVAL(K+N)
           IF(DIARY) THEN
              WRITE(4,21) EVAL(K+N)
              WRITE(4,*) ' '
              WRITE(4,*) 'FOR THE CORRESPONDING EIGENVECTOR'
           END IF
           WRITE(*,*) ' '
           WRITE(*,*) 'FOR THE CORRESPONDING EIGENVECTOR'
           DO 100 I = 1,N
               WRITE(*,34)
               WRITE(*,35) I
               IF(DIARY) WRITE(4,34)
               IF(DIARY) WRITE(4,35) I
34             FORMAT(1X,'THE REAL PART OF THE DESIRED EIGENVECTOR')
35             FORMAT(1X,'ELEMENT NUMBER',I2,' = ?')
               READ(*,*) EVEC(I,K)
               IF(DIARY) THEN
```

```
                  WRITE(4,21) EVEC(I,K)
                  WRITE(4,*) 'THE IMAGINARY PART = ?'
               END IF
               WRITE(*,*) 'THE IMAGINARY PART = ?'
               READ(*,*) EVEC(I+N,K)
               IF(DIARY) WRITE(4,21) EVEC(I+N,K)
               EVEC(I,K+1)=EVEC(I,K)
               EVEC(I+N,K+1)=-EVEC(I+N,K)
 100        CONTINUE
            ELSE
             WRITE(*,105) K
             IF(DIARY) WRITE(4,105) K
 105         FORMAT(1X,'THE DESIRED EIGENVALUE NUMBER ',I2,' = ?')
             READ(*,*) EVAL(K)
             IF(DIARY) THEN
                WRITE(4,21) EVAL(K)
                WRITE(4,*) 'FOR THE CORRESPONDING EIGENVECTOR,'
             END IF
             WRITE(*,*) 'FOR THE CORRESPONDING EIGENVECTOR,'
             DO 200 I = 1,N
                  WRITE(*,115) I
                  IF(DIARY) WRITE(4,115) I
 115              FORMAT(1X,'ELEMENT NUMBER ',I2,' = ?')
                  READ(*,*) EVEC(I,K)
                  IF(DIARY) WRITE(4,21) EVEC(I,K)
 200         CONTINUE
             TTEST = -1.0*TTEST
         END IF
 300     TTEST = -1.0*TTEST
 400     CONTINUE
         GO TO 600
C
C   OPTION 2:  ENTERING DATA FROM AN EXISTING COMPUTER DATA FILE
C
 500     WRITE(*,*)'WHAT IS THE EIGENVECTOR INPUT FILE NAME?'
         READ(*,'(A)') FILEI
         IF(DIARY) THEN
            WRITE(4,*)'WHAT IS THE EIGENVECTOR INPUT FILE NAME?'
            WRITE(4,*) FILEI
         END IF
         OPEN(3,FILE=FILEI,STATUS='OLD')
         REWIND 3
         READ(3,*,ERR=510,END=520) ((EVEC(I,J),J=1,N),I=1,N2)
         GO TO 530
 510     WRITE(6,*) '***ERROR IN INPUT FILE***'
         IF(DIARY) WRITE(4,*) '***ERROR IN INPUT FILE***'
         GO TO 500
 520     WRITE(6,*) '***ERROR: END OF FILE REACHED***'
         IF(DIARY) WRITE(4,*) '***ERROR: END OF FILE REACHED***'
         GO TO 500
 530     CLOSE(3)
 540     WRITE(*,*)'WHAT IS THE EIGENVALUE INPUT FILE NAME?'
         READ(*,'(A)') FILEII
         IF(DIARY) THEN
```

```
         WRITE(4,*)'WHAT IS THE EIGENVALUE INPUT FILE NAME?'
         WRITE(4,*) FILEII
      END IF
      OPEN(3,FILE=FILEII,STATUS='OLD')
      REWIND 3
      READ(3,*,ERR=550,END=560) (EVAL(I),I=1,N2)
      GO TO 570
550   WRITE(6,*) '***ERROR IN INPUT FILE***'
      IF(DIARY) WRITE(4,*) '***ERROR IN INPUT FILE***'
      GO TO 540
560   WRITE(6,*) '***ERROR: END OF FILE REACHED***'
      IF(DIARY) WRITE(4,*) '***ERROR: END OF FILE REACHED***'
      GO TO 540
570   CLOSE(3)
600   CONTINUE
      DO 630 I = 1, N
        WRITE(*,*) ' '
        WRITE(*,*) ' '
        IF(EVAL(I+N) .EQ. 0.0) THEN
          WRITE(*,605) I,EVAL(I)
605       FORMAT(1X,'EIGENVALUE # ',I2,' IS ',F15.7)
          WRITE(*,*) ' '
          WRITE(*,*) 'AND THE ASSOCIATED EIGENVECTOR IS'
          WRITE(*,*) ' '
          DO 610 J = 1, N
            WRITE(*,*) EVEC(J,I)
610       CONTINUE
        ELSE
          WRITE(*,615) I,EVAL(I),EVAL(I+N)
615       FORMAT(1X,'EIGENVALUE # ',I2,' IS ',F15.7,' + ',F15.7,'i')
          WRITE(*,*) 'AND THE ASSOCIATED EIGENVECTOR IS'
          DO 625 J = 1, N
            WRITE(*,620) EVEC(J,I),EVEC(J+N,I)
620         FORMAT(1X,F15.7,' + ',F15.7,'i')
625       CONTINUE
        END IF
630   CONTINUE
      WRITE(*,*) ' '
      WRITE(*,*) 'IS THIS THE DESIRED EIGENSTRUCTURE?'
      IF(DIARY) THEN
      DO 650 I = 1, N
        WRITE(4,*) ' '
        WRITE(4,*) ' '
        IF(EVAL(I+N) .EQ. 0.0) THEN
          WRITE(4,605) I,EVAL(I)
          WRITE(4,*) ' '
          WRITE(4,*) 'AND THE ASSOCIATED EIGENVECTOR IS'
          WRITE(4,*) ' '
          DO 635 J = 1, N
            WRITE(4,*) EVEC(J,I)
635       CONTINUE
        ELSE
          WRITE(4,615) I,EVAL(I),EVAL(I+N)
          WRITE(4,*) 'AND THE ASSOCIATED EIGENVECTOR IS'
```

```
         DO 640 J = 1, N
            WRITE(4,620) EVEC(J,I),EVEC(J+N,I)
640      CONTINUE
       END IF
650   CONTINUE
      WRITE(4,*) ' '
      WRITE(4,*) 'IS THIS THE DESIRED EIGENSTRUCTURE?'
      END IF
      CALL YESORN(ANSWER,DIARY)
      IF (ANSWER) GO TO 999
655   WRITE(*,*) 'DO YOU WANT TO EDIT THIS EIGENSTRUCTURE,'
      WRITE(*,*) 'OR ENTER A NEW MATRIX FROM A COMPUTER FILE?'
      IF(DIARY) WRITE(4,*) 'DO YOU WANT TO EDIT THIS EIGENSTRUCTURE,'
      IF(DIARY) WRITE(4,*)'OR ENTER A NEW MATRIX FROM A COMPUTER FILE?'
660   WRITE(*,*) 'ENTER:    E TO EDIT      OR      C FOR COMPUTER'
      READ(*,'(A1)') TEST
      IF(DIARY) THEN
      WRITE(4,*) 'ENTER:    E TO EDIT      OR      C FOR COMPUTER'
      WRITE(4,*) TEST
      END IF
      KBRD = ((LLE(TEST,'E') .AND. LGE(TEST,'E')) .OR.
     +(LLE(TEST,'e') .AND. LGE(TEST,'e')))
      IF (KBRD) GO TO 700
      IF((LLT(TEST,'C') .OR. LGT(TEST,'C')) .AND.
     +(LLT(TEST,'c') .OR. LGT(TEST,'c'))) GO TO 660
      GO TO 500
700   WRITE(*,*) ' '
      WRITE(*,*) 'WHEN YOU EDIT A COMPLEX EIGENVALUE/VECTOR, YOU'
      WRITE(*,*) 'MUST ALSO EDIT THE CONJUGATE.'
      WRITE(*,*) 'WHICH EIGENVALUE/VECTOR HAS AN INCORRECT ENTRY?'
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) 'WHEN YOU EDIT A COMPLEX EIGENVALUE/VECTOR, YOU'
         WRITE(4,*) 'MUST ALSO EDIT THE CONJUGATE.'
         WRITE(4,*) 'WHICH EIGENVALUE/VF  )R HAS AN INCORRECT ENTRY?'
      END IF
      READ(*,710) KK
710   FORMAT(I2)
      IF(DIARY) THEN
      WRITE(4,*) KK
      IF(EVAL(KK+N) .EQ. 0.0) THEN
        WRITE(4,605) KK,EVAL(KK)
        WRITE(4,*) ' '
        WRITE(4,*) 'AND THE ASSOCIATED EIGENVECTOR IS'
        WRITE(4,*) ' '
        DO 720 J = 1, N
           WRITE(4,*) EVEC(J,KK)
720     CONTINUE
      ELSE
        WRITE(4,615) KK,EVAL(KK),EVAL(KK+N)
        WRITE(4,*) 'AND THE ASSOCIATED EIGENVECTOR IS'
        DO 730 J = 1, N
           WRITE(4,620) EVEC(J,KK),EVEC(J+N,KK)
730     CONTINUE
```

```
      END IF
      END IF
      IF(EVAL(KK+N) .EQ. 0.0) THEN
        WRITE(*,605) KK,EVAL(KK)
        WRITE(*,*) ' '
        WRITE(*,*) 'AND THE ASSOCIATED EIGENVECTOR IS'
        WRITE(*,*) ' '
        DO 740 J = 1, N
           WRITE(*,*) EVEC(J,KK)
740     CONTINUE
      ELSE
        WRITE(*,615) KK,EVAL(KK),EVAL(KK+N)
        WRITE(*,*) 'AND THE ASSOCIATED EIGENVECTOR IS'
        DO 750 J = 1, N
           WRITE(*,620) EVEC(J,KK),EVEC(J+N,KK)
750     CONTINUE
      END IF
      IF(EVAL(KK+N) .NE. 0.0) THEN
         WRITE(*,16)
         WRITE(*,17) KK
         IF(DIARY) WRITE(4,16)
         IF(DIARY) WRITE(4,17) KK
         READ(*,*) EVAL(KK)
         IF(DIARY) THEN
            WRITE(4,21) EVAL(KK)
            WRITE(4,*) 'THE IMAGINARY PART = ?'
         END IF
         WRITE(*,*) 'THE IMAGINARY PART = ?'
         READ(*,*) EVAL(KK+N)
         IF(DIARY) THEN
            WRITE(4,21) EVAL(KK+N)
            WRITE(4,*) ' '
            WRITE(4,*) 'FOR THE CORRESPONDING EIGENVECTOR'
         END IF
         WRITE(*,*) ' '
         WRITE(*,*) 'FOR THE CORRESPONDING EIGENVECTOR'
         DO 760 I = 1,N
             WRITE(*,34)
             WRITE(*,35) I
             IF(DIARY) WRITE(4,34)
             IF(DIARY) WRITE(4,35) I
             READ(*,*) EVEC(I,KK)
             IF(DIARY) THEN
                WRITE(4,21) EVEC(I,KK)
                WRITE(4,*) 'THE IMAGINARY PART = ?'
             END IF
             WRITE(*,*) 'THE IMAGINARY PART = ?'
             READ(*,*) EVEC(I+N,KK)
             IF(DIARY) WRITE(4,21) EVEC(I+N,KK)
760       CONTINUE
        ELSE
         WRITE(*,105) KK
         IF(DIARY) WRITE(4,105) KK
         READ(*,*) EVAL(KK)
```

```
            IF(DIARY) THEN
               WRITE(4,21) EVAL(KK)
               WRITE(4,*) 'FOR THE CORRESPONDING EIGENVECTOR,'
            END IF
            WRITE(*,*) 'FOR THE CORRESPONDING EIGENVECTOR,'
            DO 770 I = 1,N
                 WRITE(*,115) I
                 IF(DIARY) WRITE(4,115) I
                 READ(*,*) EVEC(I,KK)
                 IF(DIARY) WRITE(4,21) EVEC(I,KK)
 770        CONTINUE
          END IF
          IF(DIARY) THEN
            WRITE(4,*) ' '
            WRITE(4,*) 'NOW'
            WRITE(4,*) ' '
            WRITE(4,*) ' '
          END IF
          WRITE(*,*) ' '
          WRITE(*,*) 'NOW'
          WRITE(*,*) ' '
          WRITE(*,*) ' '
          GO TO 600
 999      WRITE(*,*) ' '
          WRITE(*,*) ' '
          IF(DIARY) THEN
             WRITE(4,*) ' '
             WRITE(4,*) ' '
          END IF
          RETURN
          END
C*************************************************************************
          SUBROUTINE EIGV(A,Z,WR,WI,NMAX,N,DIARY)
C                                                                        |
C         CALCULATES THE EIGENSTRUCTURE OF A                             |
C*************************************************************************
          REAL A(NMAX,N),Z(NMAX,N),WR(N),WI(N),SCALE(15),H(15,15)
          INTEGER INT(15)
          LOGICAL DIARY
          CALL COPYAB(NMAX,N,N,A,H)
          CALL BALANC(NMAX,N,H,LOW,IGH,SCALE)
          CALL ELMHES(NMAX,N,LOW,IGH,H,INT)
          CALL ELTRAN(NMAX,N,LOW,IGH,H,INT,Z)
          CALL HQR2(NMAX,N,LOW,IGH,H,WR,WI,Z,IERR)
          CALL BALBAK(NMAX,N,LOW,IGH,SCALE,N,Z)
          IF(IERR .NE. 0) THEN
             WRITE(*,*) '***WARNING***WARNING***WARNING***WARNING***'
             WRITE(*,*) 'NO CONVERGENCE REACHED WHEN CALCULATING THE EIGEN
         +STRUCTURE'
             WRITE(*,*) '***WARNING***WARNING***WARNING***WARNING***'
             IF(DIARY) THEN
             WRITE(4,*) '***WARNING***WARNING***WARNING***WARNING***'
             WRITE(4,*) 'NO CONVERGENCE REACHED WHEN CALCULATING THE EIGEN
         +STRUCTURE'
```

```
         WRITE(4,*) '***WARNING***WARNING***WARNING***WARNING***'
         END IF
      END IF
      RETURN
      END
C*****************************************************************
      SUBROUTINE ELMHES (NM,N,LOW,IGH,A,INT)
C                                                                 |
C     TO REDUCE A REAL GENERAL MATRIX TO UPPER HESSENBERG FORM    |
C               USING ELEMENTARY TRANSFORMATIONS                  |
C*****************************************************************
      INTEGER I,J,M,N,LA,NM,IGH,KP1,LOW,MM1,MP1
      REAL A(NM,N)
      REAL X,Y
      REAL ABS
      INTEGER INT(IGH)
C
      LA = IGH - 1
      KP1 = LOW + 1
      IF (LA .LT. KP1) GO TO 200
C
      DO 180 M = KP1, LA
         MM1 = M - 1
         X = 0.0
         I = M
C
         DO 100 J = M, IGH
            IF (ABS(A(J,MM1)) .LE. ABS(X)) GO TO 100
            X = A(J,MM1)
            I = J
 100     CONTINUE
C
         INT(M) = I
         IF (I .EQ. M) GO TO 130
C     ********** INTERCHANGE ROWS AND COLUMNS OF A **********
         DO 110 J = MM1, N
            Y = A(I,J)
            A(I,J) = A(M,J)
            A(M,J) = Y
 110     CONTINUE
C
         DO 120 J = 1, IGH
            Y = A(J,I)
            A(J,I) = A(J,M)
            A(J,M) = Y
 120     CONTINUE
C     ********** END INTERCHANGE **********
 130     IF (X .EQ. 0.0) GO TO 180
         MP1 = M + 1
C
         DO 160 I = MP1, IGH
         Y = A(I,MM1)
         IF (Y .EQ. 0.0) GO TO 160
         Y = Y / X
```

```
         A(I,MM1) = Y
C
         DO 140 J = M, N
 140     A(I,J) = A(I,J) - Y * A(M,J)
C
         DO 150 J = 1, IGH
 150     A(J,M) = A(J,M) + Y * A(J,I)
C
 160  CONTINUE
C
 180  CONTINUE
C
 200  RETURN
      END
C*********************************************************************|
      SUBROUTINE ELTRAN(NM,N,LOW,IGH,A,INT,Z)
C                                                                     |
C     TO ACCUMULATE THE TRANSFORMATIONS IN THE REDUCTION OF A REAL    |
C                     GENERAL MATRIX BY ELMHES                        |
C*********************************************************************|
      INTEGER I,J,N,KL,MM,MP,NM,IGH,LOW,MP1
      REAL A(NM,IGH),Z(NM,N)
      INTEGER INT(IGH)
C
C     ********** INITIALIZE Z TO IDENTITY MATRIX **********
      DO 80 I = 1, N
C
         DO 60 J = 1, N
  60     Z(I,J) = 0.0
C
         Z(I,I) = 1.0
  80  CONTINUE
C
      KL = IGH - LOW - 1
      IF (KL .LT. 1) GO TO 200
C     ********** FOR MP=IGH-1 STEP -1 UNTIL LOW+1 DO -- **********
      DO 140 MM = 1, KL
         MP = IGH - MM
         MP1 = MP + 1
C
      DO 100 I = MP1, IGH
 100  Z(I,MP) = A(I,MP-1)
C
      I = INT(MP)
      IF (I .EQ. MP) GO TO 140
C
      DO 130 J = MP, IGH
         Z(MP,J) = Z(I,J)
         Z(I,J) = 0.0
 130  CONTINUE
C
         Z(I,MP) = 1.0
 140  CONTINUE
C
```

```
 200  RETURN
      END
C*****************************************************************|
      SUBROUTINE FACTOR(A,W,IPIVOT,D,N,IFLAG,NMAX)
C                                                                 |
C     USED TO INVERT A MATRIX WITH SUBROUTINE SUBST               |
C*****************************************************************|
      DIMENSION A(NMAX,N),W(NMAX,N),IPIVOT(N),D(N)
      IFLAG = 1
C
C     INITIALIZE W, IPIVOT, AND D
C
      DO 10 I = 1, N
         IPIVOT(I) = I
         ROWMAX = 0.
         DO 9 J = 1, N
            ROWMAX = AMAX1(ROWMAX,ABS(W(I,J)))
  9      CONTINUE
         IF (ROWMAX .EQ. 0.) GO TO 999
         D(I) = ROWMAX
 10   CONTINUE
C
C     GAUSS ELIMINATION WITH SCALED PARTIAL PIVOTING
C
      NM1 = N - 1
      IF (NM1 .EQ. 0) RETURN
      DO 20 K = 1, NM1
         J = K
         KP1 = K+1
         IP = IPIVOT(K)
         COLMAX = ABS(W(IP,K))/D(IP)
         DO 11 I = KP1, N
            IP = IPIVOT(I)
            AWIKOV = ABS(W(IP,K))/D(IP)
            IF (AWIKOV .LE. COLMAX) GO TO 11
            COLMAX = AWIKOV
            J = I
 11      CONTINUE
         IF (COLMAX .EQ. 0.) GO TO 999
         IPK = IPIVOT(J)
         IPIVOT(J) = IPIVOT(K)
         IPIVOT(K) = IPK
         DO 20 I = KP1, N
            IP = IPIVOT(I)
            W(IP,K) = W(IP,K)/W(IPK,K)
            RATIO = -W(IP,K)
            DO 20 J = KP1, N
               W(IP,J) = RATIO * W(IPK,J) + W(IP,J)
 20   CONTINUE
      IF (W(IP,N) .EQ. 0.) GO TO 999
      RETURN
 999  IFLAG = 2
      RETURN
      END
```

```
C*****************************************************************|
      SUBROUTINE HQR2(NM,N,LOW,IGH,H,WR,WI,Z,IERR)
C                                                                 |
C     COMPUTES THE EIGENVALUS AND EIGENVECTORS OF A REAL UPPER    |
C                        HESSENBERG MATRIX                        |
C*****************************************************************|
      INTEGER I,J,K,L,M,N,EN,II,JJ,LL,MM,NA,NM,NN,
     X        IGH,ITS,LOW,MP2,ENM2,IERR
      REAL H(NM,N),WR(N),WI(N),Z(NM,N)
      REAL P,Q,R,S,T,W,X,Y,RA,SA,VI,VR,ZZ,NORM,MACHEP
      REAL SQRT,ABS,SIGN
      INTEGER MINO
      LOGICAL NOTLAS
      COMPLEX Z3
      COMPLEX CMPLX
      REAL REAL,AIMAG
C
C     ********** MACHEP IS A MACHINE DEPENDENT PARAMETER SPECIFYING
C                THE RELATIVE PRECISION OF FLOATING POINT ARITHMETIC.
C
C                **********
      MACHEP = 1.E-8
C
      IERR = 0
      NORM = 0.0
      K = 1
C     ********** STORE ROOTS ISOLATED BY BALANC
C                AND COMPUTE MATRIX NORM **********
      DO 50 I = 1, N
C
         DO 40 J = K, N
  40     NORM = NORM + ABS(H(I,J))
C
         K = I
         IF (I .GE. LOW .AND. I .LE. IGH) GO TO 50
         WR(I) = H(I,I)
         WI(I) = 0.0
  50  CONTINUE
C
      EN = IGH
      T = 0.0
C     ********** SEARCH FOR NEXT EIGENVALUES **********
  60  IF (EN .LT. LOW) GO TO 340
      ITS = 0
      NA = EN - 1
      ENM2 = NA - 1
C     ********** LOOK FOR SINGLE SMALL SUB-DIAGONAL ELEMENT
C                        FOR L=EN STEP -1 UNTIL LOW DO -- **********
  70  DO 80 LL = LOW, EN
         L = EN + LOW - LL
         IF (L .EQ. LOW) GO TO 100
         S = ABS(H(L-1,L-1)) + ABS(H(L,L))
         IF (S .EQ. 0.0) S = NORM
         IF (ABS(H(L,L-1)) .LE. MACHEP * S) GO TO 100
```

```
   80 CONTINUE
C     ********** FORM SHIFT **********
  100 X = H(EN,EN)
      IF (L .EQ. EN) GO TO 270
      Y = H(NA,NA)
      W = H(EN,NA) * H(NA,EN)
      IF (L .EQ. NA) GO TO 280
      IF (ITS .EQ. 30) GO TO 1000
      IF (ITS .NE. 10 .AND. ITS .NE. 20) GO TO 130
C     ********** FORM EXCEPTIONAL SHIFT **********
      T = T + X
C
      DO 120 I = LOW, EN
  120 H(I,I) = H(I,I) - X
C
      S = ABS(H(EN,NA)) + ABS(H(NA,ENM2))
      X = 0.75 * S
      Y = X
      W = -0.4375 * S * S
  130 ITS = ITS + 1
C     ********** LOOK FOR TWO CONSECUTIVE SMALL
C                SUB-DIAGONAL ELEMENTS.
C                FOR M=EN-2 STEP -1 UNTIL L DO -- **********
      DO 140 MM = L, ENM2
         M = ENM2 + L - MM
         ZZ = H(M,M)
         R = X - ZZ
         S = Y - ZZ
         P = (R * S - W) / H(M+1,M) + H(M,M+1)
         Q = H(M+1,M+1) - ZZ - R - S
         R = H(M+2,M+1)
         S = ABS(P) + ABS(Q) + ABS(R)
         P = P / S
         Q = Q / S
         R = R / S
         IF (M .EQ. L) GO TO 150
         IF (ABS(H(M,M-1)) * (ABS(Q) + ABS(R)) .LE. MACHEP * ABS(P)
     X    * (ABS(H(M-1,M-1)) + ABS(ZZ) + ABS(H(M+1,M+1)))) GO TO 150
  140 CONTINUE
C
  150 MP2 = M + 2
C
      DO 160 I = MP2, EN
         H(I,I-2) = 0.0
         IF (I .EQ. MP2) GO TO 160
         H(I,I-3) = 0.0
  160 CONTINUE
C     ********** DOUBLE QR STEP INVOLVING ROWS L TO EN AND
C                COLUMNS M TO EN **********
      DO 260 K = M, NA
         NOTLAS = K .NE. NA
         IF (K .EQ. M) GO TO 170
         P = H(K,K-1)
         Q = H(K+1,K-1)
```

```
            R = 0.0
            IF (NOTLAS) R = H(K+2,K-1)
            X = ABS(P) + ABS(Q) + ABS(R)
            IF (X .EQ. 0.0) GO TO 260
            P = P / X
            Q = Q / X
            R = R / X
  170       S = SIGN(SQRT(P*P+Q*Q+R*R),P)
            IF (K .EQ. M) GO TO 180
            H(K,K-1) = -S * X
            GO TO 190
  180       IF (L .NE. M) H(K,K-1) = -H(K,K-1)
  190       P = P + S
            X = P / S
            Y = Q / S
            ZZ = R / S
            Q = Q / P
            R = R / P
C       ********** ROW MODIFICATION **********
            DO 210 J = K, N
               P = H(K,J) + Q * H(K+1,J)
               IF (.NOT. NOTLAS) GO TO 200
               P = P + R * H(K+2,J)
               H(K+2,J) = H(K+2,J) - P * ZZ
  200          H(K+1,J) = H(K+1,J) - P * Y
               H(K,J) = H(K,J) - P * X
  210       CONTINUE
C
            J = MINO(EN,K+3)
C       ********** COLUMN MODIFICATION **********
            DO 230 I = 1, J
               P = X * H(I,K) + Y * H(I,K+1)
               IF (.NOT. NOTLAS) GO TO 220
               P = P + ZZ * H(I,K+2)
               H(I,K+2) = H(I,K+2) - P * R
  220          H(I,K+1) = H(I,K+1) - P * Q
               H(I,K) = H(I,K) - P
  230       CONTINUE
C       ********** ACCUMULATE TRANSFORMATIONS **********
            DO 250 I = LOW, IGH
               P = X * Z(I,K) + Y * Z(I,K+1)
               IF (.NOT. NOTLAS) GO TO 240
               P = P + ZZ * Z(I,K+2)
               Z(I,K+2) = Z(I,K+2) - P * R
  240          Z(I,K+1) = Z(I,K+1) - P * Q
               Z(I,K) = Z(I,K) - P
  250       CONTINUE
C
  260    CONTINUE
C
         GO TO 70
C       ********** ONE ROOT FOUND **********
  270    H(EN,EN) = X + T
         WR(EN) = H(EN,EN)
```

```
      WI(EN) = 0.0
      EN = NA
      GO TO 60
C     ********** TWO ROOTS FOUND **********
  280 P = (Y - X) / 2.0
      Q = P * P + W
      ZZ = SQRT(ABS(Q))
      H(EN,EN) = X + T
      X = H(EN,EN)
      H(NA,NA) = Y + T
      IF (Q .LT. 0.0) GO TO 320
C     ********** REAL PAIR **********
      ZZ = P + SIGN(ZZ,P)
      WR(NA) = X + ZZ
      WR(EN) = WR(NA)
      IF (ZZ .NE. 0.0) WR(EN) = X - W / ZZ
      WI(NA) = 0.0
      WI(EN) = 0.0
      X = H(EN,NA)
      S = ABS(X) + ABS(ZZ)
      P = X / S
      Q = ZZ / S
      R = SQRT(P*P+Q*Q)
      P = P / R
      Q = Q / R
C     ********** ROW MODIFICATION **********
      DO 290 J = NA, N
         ZZ = H(NA,J)
         H(NA,J) = Q * ZZ + P * H(EN,J)
         H(EN,J) = Q * H(EN,J) - P * ZZ
  290 CONTINUE
C     ********** COLUMN MODIFICATION **********
      DO 300 I = 1, EN
         ZZ = H(I,NA)
         H(I,NA) = Q * ZZ + P * H(I,EN)
         H(I,EN) = Q * H(I,EN) - P * ZZ
  300 CONTINUE
C     ********** ACCUMULATE TRANSFORMATIONS **********
      DO 310 I = LOW, IGH
         ZZ = Z(I,NA)
         Z(I,NA) = Q * ZZ + P * Z(I,EN)
         Z(I,EN) = Q * Z(I,EN) - P * ZZ
  310 CONTINUE
C
      GO TO 330
C     ********** COMPLEX PAIR **********
  320 WR(NA) = X + P
      WR(EN) = X + P
      WI(NA) = ZZ
      WI(EN) = -ZZ
  330 EN = ENM2
      GO TO 60
C     ********** ALL ROOTS FOUND.  BACKSUBSTITUE TO FIND
C                VECTORS OF UPPER TRIANGULAR FORM **********
```

```
  340 IF (NORM .EQ. 0.0) GO TO 1001
C     ********** FOR EN=N STEP -1 UNTIL 1 DO -- **********
      DO 800 NN = 1, N
         EN = N + 1 - NN
         P = WR(EN)
         Q = WI(EN)
         NA = EN - 1
         IF (Q) 710, 600, 800
C     ********** REAL VECTOR **********
  600    M = EN
         H(EN,EN) = 1.0
         IF (NA .EQ. 0) GO TO 800
C     ********** FO I=EN-1 STEP -1 UNTIL 1 DO -- **********
         DO 700 II = 1, NA
            I = EN - II
            W = H(I,I) - P
            R = H(I,EN)
            IF (M .GT. NA) GO TO 620
C
            DO 610 J = M, NA
  610       R = R + H(I,J) * H(J,EN)
C
  620       IF (WI(I) .GE. 0.0) GO TO 630
            ZZ = W
            S = R
            GO TO 700
  630       M = I
            IF (WI(I) .NE. 0.0) GO TO 640
            T = W
            IF (W .EQ. 0.0) T = MACHEP * NORM
            H(I,EN) = -R / T
            GO TO 700
C     ********** SOLVE REAL EQUATIONS **********
  640       X = H(I,I+1)
            Y = H(I+1,I)
            Q = (WR(I) - P) * (WR(I) - P) + WI(I) * WI(I)
            T = (X * S - ZZ * R) / Q
            H(I,EN) = T
            IF (ABS(X) .LE. ABS(ZZ)) GO TO 650
            H(I+1,EN) = (-R - W * T) / X
            GO TO 700
  650       H(I+1,EN) = (-S - Y * T) / ZZ
  700    CONTINUE
C     ********** END REAL VECTOR **********
         GO TO 800
C     ********** COMPLEX VECTOR **********
  710    M = NA
C     ********** LAST VECTOR COMPONENT CHOSEN IMAGINARY SO THAT
C                EIGENVECTOR MATRIX IS TRIANGULAR **********
         IF (ABS(H(EN,NA)) .LE. ABS(H(NA,EN))) GO TO 720
         H(NA,NA) = Q / H(EN,NA)
         H(NA,EN) = -(H(EN,EN) - P) / H(EN,NA)
         GO TO 730
  720    Z3 = CMPLX(0.0,-H(NA,EN)) / CMPLX(H(NA,NA)-P,Q)
```

```
            H(NA,NA) = REAL(Z3)
            H(NA,EN) = AIMAG(Z3)
  730       H(EN,NA) = 0.0
            H(EN,EN) = 1.0
            ENM2 = NA-1
            IF (ENM2 .EQ. 0) GO TO 800
C     ********** FOR I=EN-2 STEP -1 UNTIL 1 DO -- **********
            DO 790 II = 1, ENM2
               I = NA - II
               W = H(I,I) - P
               RA = 0.0
               SA = H(I,EN)
C
               DO 760 J = M, NA
                  RA = RA + H(I,J) * H(J,NA)
                  SA = SA + H(I,J) * H(J,EN)
  760          CONTINUE
C
               IF (WI(I) .GE. 0.0) GO TO 770
               ZZ = W
               R = RA
               S = SA
               GO TO 790
  770          M = I
               IF (WI(I) .NE. 0.0) GO TO 780
               Z3 = CMPLX(-RA,-SA) / CMPLX(W,Q)
               H(I,NA) = REAL(Z3)
               H(I,EN) = AIMAG(Z3)
               GO TO 790
C     ********** SOLVE COMPLEX EQUATIONS **********
  780          X = H(I,I+1)
               Y = H(I+1,I)
               VR = (WR(I) - P) * (WR(I) - P) + WI(I) * WI(I) - Q * Q
               VI = (WR(I) - P) * 2.0 * Q
               IF (VR .EQ. 0.0 .AND. VI .EQ. 0.0) VR = MACHEP * NORM
     X          * (ABS(W) + ABS(Q) + ABS(X) + ABS(Y) + ABS(ZZ))
               Z3 = CMPLX(X*R-ZZ*RA+Q*SA,X*S-ZZ*SA-Q*RA) / CMPLX(VR,VI)
               H(I,NA) = REAL(Z3)
               H(I,EN) = AIMAG(Z3)
               IF (ABS(X) .LE. ABS(ZZ) + ABS(Q)) GO TO 785
               H(I+1,NA) = (-RA - W * H(I,NA) + Q * H(I,EN)) / X
               H(I+1,EN) = (-SA - W * H(I,EN) - Q * H(I,NA)) / X
               GO TO 790
  785          Z3 = CMPLX(-R-Y*H(I,NA),-S-Y*H(I,EN)) / CMPLX(ZZ,Q)
               H(I+1,NA) = REAL(Z3)
               H(I+1,EN) = AIMAG(Z3)
  790       CONTINUE
C     ********** END COMPLEX VECTOR **********
  800 CONTINUE
C     ********** END BACK SUBSTITUTION.
C                VECTORS OF ISOLATED ROOTS **********
      DO 840 I = 1, N
         IF (I .GE. LOW .AND. I .LE. IGH) GO TO 840
C
```

```
         DO 820 J = I, N
  820    Z(I,J) = H(I,J)
C
  840 CONTINUE
C     ********** MULTIPLY BY TRANSFORMATION MATRIX TO GIVE
C                VECTORS OF ORIGINAL FULL MATRIX.
C                FOR J=N STEP -1 UNTIL LOW DO -- **********
      DO 880 JJ = LOW, N
         J = N + LOW - JJ
         M = MINO(J,IGH)
C
         DO 880 I = LOW, IGH
            ZZ = 0.0
C
            DO 860 K = LOW, M
  860       ZZ = ZZ + Z(I,K) * H(K,J)
C
            Z(I,J) = ZZ
  880 CONTINUE
C
      GO TO 1001
C     ********** SET ERROR -- NO CONVERGENCE TO AN
C                EIGENVALUE AFTER 30 ITERATIONS **********
 1000 IERR = EN
 1001 RETURN
      END
C*********************************************************************|
      SUBROUTINE INPUTA(NMAX,N,M,A,DIARY)
C                                                                     |
C     THIS SUBROUTINE READS IN ANY NXM MATRIX CALLED A                |
C     THE PARAMETERS N AND M ARE PASSED TO IT                         |
C     AND THIS SUBROUTINE PASSES BACK THE MATRIX A                    |
C*********************************************************************|
      REAL A(NMAX,M)
      LOGICAL ANSWER,KBRD,DIARY
      CHARACTER TEST*1,FILEI*14
      WRITE(*,*) ' '
      WRITE(*,*) ' '
    1 WRITE(*,*) 'DO YOU WANT TO USE A NEW MATRIX ENTERED FROM THE'
      WRITE(*,*) 'KEYBOARD, OR ONE ALREADY STORED ON THE COMPUTER?'
      WRITE(*,*) ' '
      WRITE(*,*) 'ENTER:    K FOR KEYBOARD    OR    C FOR COMPUTER'
      IF(DIARY) THEN
      WRITE(4,*) ' '
      WRITE(4,*) ' '
      WRITE(4,*) 'DO YOU WANT TO USE A NEW MATRIX ENTERED FROM THE'
      WRITE(4,*) 'KEYBOARD, OR ONE ALREADY STORED ON THE COMPUTER?'
      WRITE(4,*) ' '
      WRITE(4,*) 'ENTER:    K FOR KEYBOARD    OR    C FOR COMPUTER'
      END IF
      READ(*,'(A1)') TEST
      IF(DIARY) WRITE(4,*) TEST
      KBRD = ((LLE(TEST,'K') .AND. LGE(TEST,'K')) .OR.
     +(LLE(TEST,'k') .AND. LGE(TEST,'k')))
```

```
      IF (KBRD) GO TO 2
      IF((LLT(TEST,'C') .OR. LGT(TEST,'C')) .AND.
     +(LLT(TEST,'c') .OR. LGT(TEST,'c'))) GO TO 1
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
         WRITE(4,*) ' '
      END IF
      GO TO 200
C
C   OPTION 1:  ENTERING DATA FROM THE KEYBOARD
C
  2   WRITE(*,*) ' '
      WRITE(*,*) ' '
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
      END IF
      DO 30 I=1,N
          DO 20 J=1,M
            WRITE(*,3) I,J
            IF(DIARY) WRITE(4,3) I,J
  3         FORMAT(1X,'ENTER THE ELEMENT OF ROW ',I2,' AND COLUMN ',I2)
            READ(*,*) A(I,J)
            IF(DIARY) WRITE(4,5) A(I,J)
  5         FORMAT(1X,F15.7)
 20       CONTINUE
 30   CONTINUE
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
      END IF
      WRITE(*,*) ' '
      WRITE(*,*) ' '
 40   WRITE(*,*) 'THE MATRIX IS GIVEN AS'
      IF(DIARY) WRITE(4,*) 'THE MATRIX IS GIVEN AS'
      DO 60 I=1,N
          WRITE(*,50) (A(I,J),J=1,M)
          WRITE(*,*) ' '
          IF(DIARY) THEN
             WRITE(4,50) (A(I,J),J=1,M)
             WRITE(4,*) ' '
          END IF
 50       FORMAT(5F15.7)
 60   CONTINUE
      WRITE(*,*) ' '
      WRITE(*,*) 'IS THIS MATRIX CORRECT?'
      IF(DIARY) THEN
      WRITE(4,*) ' '
      WRITE(4,*) 'IS THIS MATRIX CORRECT?'
      END IF
```

```
      CALL YESORN(ANSWER,DIARY)
      IF(ANSWER) GO TO 999
 70   WRITE(*,*) 'WHICH ROW HAS AN INCORRECT ENTRY?'
      IF(DIARY) WRITE(4,*) 'WHICH ROW HAS AN INCORRECT ENTRY?'
      READ(*,80) K
 80   FORMAT(I2)
      IF(DIARY) THEN
         WRITE(4,*) K
         WRITE(4,*) 'WHICH COLUMN HAS AN INCORRECT ENTRY?'
      END IF
      WRITE(*,*) 'WHICH COLUMN HAS AN INCORRECT ENTRY?'
      READ(*,90) L
 90   FORMAT(I2)
      IF(DIARY) THEN
      WRITE(4,*) L
      WRITE(4,*) 'THE CURRENT VALUE OF A(',K,',',L,') IS ',A(K,L)
      WRITE(4,*) 'ENTER THE CORRECT VALUE'
      END IF
      WRITE(*,*) 'THE CURRENT VALUE OF A(',K,',',L,') IS ',A(K,L)
      WRITE(*,*) 'ENTER THE CORRECT VALUE'
      READ(*,*) A(K,L)
      IF(DIARY) THEN
         WRITE(4,*) A(K,L)
         WRITE(4,*) ' '
         WRITE(4,*) 'NOW'
         WRITE(4,*) ' '
         WRITE(4,*) ' '
      END IF
      WRITE(*,*) ' '
      WRITE(*,*) 'NOW'
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      GO TO 40
C
C   OPTION 2:  ENTERING DATA FROM AN EXISTING COMPUTER DATA FILE
C
 200  WRITE(*,*)'WHAT IS THE INPUT FILE NAME?'
      IF(DIARY) WRITE(4,*)'WHAT IS THE INPUT FILE NAME?'
      READ(*,'(A)') FILEI
      IF(DIARY) WRITE(4,*) FILEI
      OPEN(3,FILE=FILEI,STATUS='OLD')
      REWIND 3
      READ(3,*,ERR=210,END=220) ((A(I,J),J=1,M),I=1,N)
      GO TO 230
 210  WRITE(6,*) '***ERROR IN INPUT FILE***'
      IF(DIARY) WRITE(4,*) '***ERROR IN INPUT FILE***'
      GO TO 200
 220  WRITE(6,*) '***ERROR: END OF FILE REACHED***'
      IF(DIARY) WRITE(4,*) '***ERROR: END OF FILE REACHED***'
      GO TO 200
 230  CLOSE(3)
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) 'THE MATRIX READ IS'
```

```
      WRITE(*,*) ' '
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
         WRITE(4,*) 'THE MATRIX READ IS'
         WRITE(4,*) ' '
      END IF
      DO 250 I=1,N
          WRITE(*,240) (A(I,J),J=1,M)
          WRITE(*,*) ' '
          IF(DIARY) THEN
             WRITE(4,240) (A(I,J),J=1,M)
             WRITE(4,*) ' '
          END IF
 240      FORMAT(5F15.7)
 250  CONTINUE
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) 'IS THIS THE DESIRED MATRIX?'
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
         WRITE(4,*) 'IS THIS THE DESIRED MATRIX?'
      END IF
      CALL YESORN(ANSWER,DIARY)
      IF (ANSWER) GO TO 999
      WRITE(*,*) 'DO YOU WANT TO EDIT THIS MATRIX,'
      WRITE(*,*) 'OR ENTER A NEW MATRIX?'
      IF(DIARY) WRITE(4,*) 'DO YOU WANT TO EDIT THIS MATRIX,'
      IF(DIARY) WRITE(4,*) 'OR ENTER A NEW MATRIX?'
 260  WRITE(*,*) 'ENTER:    E TO EDIT      OR      N FOR NEW'
      READ(*,'(A1)') TEST
      IF(DIARY) THEN
      WRITE(4,*) 'ENTER:    E TO EDIT      OR      N FOR NEW'
      WRITE(4,*) TEST
      END IF
      KBRD = LLE(TEST,'E') .AND. LGE(TEST,'E')
      IF (KBRD) GO TO 70
      IF(LLT(TEST,'N') .OR. LGT(TEST,'N')) GO TO 260
      GO TO 1
 999  WRITE(*,*) ' '
      WRITE(*,*) ' '
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
      END IF
      RETURN
      END
C*********************************************************************|
      SUBROUTINE INVERT(A,AINV,N,DIARY)
C                                                                     |
C     CALLS SUBROUTINES FACTOR AND SUBST TO INVERT A MATRIX           |
C*********************************************************************|
      REAL AA(15,15),AINV(15,N),B(15),A(15,N)
```

```
      INTEGER IPIVOT(15)
      LOGICAL DIARY
      DO 10 I = 1, N
         DO 10 J = 1, N
            AA(I,J) = A(I,J)
   10 CONTINUE
      CALL FACTOR(AA,AA,IPIVOT,B,N,IFLAG,15)
      GO TO (30,20)IFLAG
   20 WRITE(*,*) 'MATRIX IS SINGULAR'
      IF(DIARY) WRITE(4,*) 'MATRIX IS SINGULAR'
      RETURN
   30 DO 40 I = 1, N
      B(I) = 0.
   40 CONTINUE
      DO 50 J = 1, N
         B(J) = 1.
         CALL SUBST(AA,B,AINV(1,J),IPIVOT,N,15)
         B(J) = 0.
   50 CONTINUE
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
      END IF
      RETURN
      END
C*************************************************************************|
      SUBROUTINE PACHDF(PA,PACH,NMAX,N,M,K)
C                                                                         |
C     DEFINES PACH FROM PA                                                |
C*************************************************************************|
      REAL PACH(NMAX,M),PA(NMAX)
      DO 10 I = 1, N
         PACH(I,K) = PA(I)
   10 CONTINUE
      RETURN
      END
C*************************************************************************|
      SUBROUTINE PDDEF(EVEC,PD,NMAX,N,M,K)
C                                                                         |
C     DEFINES THE DESIRED EIGENVECTOR PD FROM THE MATRIX EVEC             |
C*************************************************************************|
      REAL EVEC(NMAX,M),PD(NMAX)
      DO 10 I = 1, N
         PD(I) = EVEC(I,K)
   10 CONTINUE
      RETURN
      END
C*************************************************************************|
      SUBROUTINE PLOT(XHISTR,KP1,N,T,NMAX,DIARY)
C                                                                         |
C     PLOTTING ROUTINE USING INPUTS FROM SUBROUTINE SIMUL8                |
C*************************************************************************|
```

```
      REAL XMAX(15),XMIN(15),XHISTR(15,151)
      LOGICAL ANSWER,DIARY
      CHARACTER*1 BLANK,PLUS,STAR
      BLANK = ' '
      PLUS = '+'
      STAR = '*'
C
C     XMAX(I) IS THE LARGEST VALUE OF X(I).
C     XMIN(I) IS THE SMALLEST VALUE OF X(I).
C     THESE ARE USED TO SCALE THE PLOT OF THE TIME HISTORY OF X(I).
C
      DO 10 I = 1, N
         XMAX(I) = XHISTR(I,1)
         XMIN(I) = XHISTR(I,1)
   10 CONTINUE
      DO 20 I = 1, N
         DO 20 J = 1, KP1
            XMAX(I) = AMAX1(XHISTR(I,J),XMAX(I))
            XMIN(I) = AMIN1(XHISTR(I,J),XMIN(I))
   20 CONTINUE
   30 WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) 'WHICH ELEMENT OF THE STATE VECTOR X'
      WRITE(*,*) 'DO YOU WANT PLOTTED?'
      WRITE(*,*) 'ENTER THE DESIRED VALUE OF I, I = 1, 2,..., N,'
      WRITE(*,*) 'FOR X(I).'
      READ(*,'(I2)') I
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
         WRITE(4,*) ' '
         WRITE(4,*) 'WHICH ELEMENT OF THE STATE VECTOR X'
         WRITE(4,*) 'DO YOU WANT PLOTTED?'
         WRITE(4,*) 'ENTER THE DESIRED VALUE OF I, I = 1, 2,..., N,'
         WRITE(4,*) 'FOR X(I).'
         WRITE(4,*) I
         WRITE(4,40) I,XMIN(I)
         WRITE(4,50) I,XMAX(I)
         WRITE(4,*) ' '
         WRITE(4,*) 'NOTE THAT THE SYMBOL "*" IS THE PLOTTED VALUE'
         WRITE(4,*) 'AND THE SYMBOL "+" SHOWS THE LOCATION OF ZERO'
         WRITE(4,*) ' '
      END IF
      WRITE(*,40) I,XMIN(I)
   40 FORMAT(1X,'XMIN(',I2,')=',E12.5)
      WRITE(*,50) I,XMAX(I)
   50 FORMAT(1X,'XMAX(',I2,')=',E12.5)
      WRITE(*,*) ' '
      WRITE(*,*) 'NOTE THAT THE SYMBOL "*" IS THE PLOTTED VALUE'
      WRITE(*,*) 'AND THE SYMBOL "+" SHOWS THE LOCATION OF ZERO'
      WRITE(*,*) ' '
      DO 60 KT = 1, KP1
       TIME = (KT-1) * T
```

```
       IF (XMAX(I).GT.0.0) THEN
          IF (XMIN(I).LT.0.0) THEN
C*******************
C    CASE I (XMIN < 0 < XMAX)
             NSTAR = NINT(69.0*(XHISTR(I,KT)-XMIN(I))/(XMAX(I)-
     +XMIN(I)))+1
             NPLUS = NINT(69.0*(0.0-XMIN(I))/(XMAX(I)-XMIN(I)))+1
C*******************
             IF (NSTAR.LT.NPLUS) THEN
                WRITE(*,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NSTAR-1),
     +STAR,(BLANK,L=1,NPLUS-NSTAR-1),PLUS
              IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NSTAR-1),
     +STAR,(BLANK,L=1,NPLUS-NSTAR-1),PLUS
             END IF
             IF (NSTAR.EQ.NPLUS) THEN
                WRITE(*,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NSTAR-1),
     +STAR
              IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NSTAR-1),
     +STAR
             END IF
             IF (NSTAR.GT.NPLUS) THEN
                WRITE(*,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NPLUS-1),
     +PLUS,(BLANK,L=1,NSTAR-NPLUS-1),STAR
              IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NPLUS-1),
     +PLUS,(BLANK,L=1,NSTAR-NPLUS-1),STAR
             END IF
          END IF
          IF (XMIN(I).EQ.0.0) THEN
C*******************
C    CASE II (0 = XMIN < XMAX)
             NSTAR = NINT(69.0*(XHISTR(I,KT)-XMIN(I))/(XMAX(I)-
     +XMIN(I)))+1
             NPLUS = 1
C*******************
             IF (NSTAR.EQ.NPLUS) THEN
                WRITE(*,'('' '',F7.3,70A)')TIME,STAR
                IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,STAR
             ELSE
                WRITE(*,'('' '',F7.3,70A)')TIME,PLUS,(BLANK,L=1,
     +NSTAR-NPLUS-1),STAR
              IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,PLUS,(BLANK,L=1,
     +NSTAR-NPLUS-1),STAR
             END IF
          END IF
          IF (XMIN(I).GT.0.0) THEN
C*******************
                IF (XMIN(I).EQ.XMAX(I)) THEN
C   CASE VII (0 < XMIN = XMAX)
                   NSTAR = 35
                ELSE
C   CASE VI (0 < XMIN < XMAX)
                   NSTAR = NINT(69.0*(XHISTR(I,KT)/XMAX(I)))+1
                END IF
C*******************
```

```
                  WRITE(*,'('' '',F7.3,70A)')TIME,PLUS,(BLANK,L=1,
     +NSTAR-2),STAR
                  IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,PLUS,(BLANK,L=1,
     +NSTAR-2),STAR
               END IF
            END IF
            IF (XMAX(I).EQ.0.0) THEN
               IF (XMIN(I).LT.XMAX(I)) THEN
C*******************
C     CASE III (XMIN < 0 = XMAX)
                  NSTAR = NINT(69.0*(XHISTR(I,KT)-XMIN(I))/(XMAX(I)-
     +XMIN(I)))+1
                  NPLUS = 70
C*******************
                  IF (NSTAR.EQ.NPLUS) THEN
                     WRITE(*,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NSTAR-1),
     +STAR
                  IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NSTAR-1),
     +STAR
                  ELSE
                     WRITE(*,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NSTAR-1),
     +STAR,(BLANK,L=1,NPLUS-NSTAR-1),PLUS
                  IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NSTAR-1),
     +STAR,(BLANK,L=1,NPLUS-NSTAR-1),PLUS
                  END IF
               ELSE
C*******************
C     CASE IV (0 = XMIN = XMAX)
C*******************
                  WRITE(*,'('' '',F7.3,70A)')TIME,(BLANK,L=1,35),STAR
            IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,(BLANK,L=1,35),STAR
               END IF
            END IF
            IF (XMAX(I).LT.0.0) THEN
               IF (XMIN(I).EQ.XMAX(I)) THEN
C*******************
C     CASE VIII (XMIN = XMAX < 0)
C*******************
                  WRITE(*,'('' '',F7.3,70A)')TIME,(BLANK,L=1,10),
     +STAR,(BLANK,L=1,25),PLUS
                  IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,(BLANK,L=1,10),
     +STAR,(BLANK,L=1,25),PLUS
               ELSE
C*******************
C     CASE V (XMIN < XMAX < 0)
                  NSTAR = NINT(69.0*(XHISTR(I,KT)-XMIN(I))/(0.0-
     +XMIN(I)))+1
                  NPLUS = 70
C*******************
                  WRITE(*,'('' '',F7.3,70A)')TIME,(BLANK,L=1,NSTAR-1),
     +STAR,(BLANK,L=1,NPLUS-NSTAR-1),PLUS
                  IF(DIARY) WRITE(4,'('' '',F7.3,70A)')TIME,
     +(BLANK,L=1,NSTAR-1),STAR,(BLANK,L=1,NPLUS-NSTAR-1),PLUS
               END IF
```

```
      END IF
 60   CONTINUE
      IF(DIARY) THEN
         WRITE(4,70) I,XMIN(I)
         WRITE(4,80) I,XMAX(I)
         WRITE(4,*) ' '
         WRITE(4,*) ' '
         WRITE(4,*) ' '
         WRITE(4,*) 'DO YOU WANT TO PLOT ANOTHER ELEMENT OF THE'
         WRITE(4,*) 'STATE VECTOR X?'
      END IF
      WRITE(*,70) I,XMIN(I)
  70  FORMAT(1X,'XMIN(',I2,')=',E12.5)
      WRITE(*,80) I,XMAX(I)
  80  FORMAT(1X,'XMAX(',I2,')=',E12.5)
C     END OF THE PLOTTING SUBROUTINE
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) 'DO YOU WANT TO PLOT ANOTHER ELEMENT OF THE'
      WRITE(*,*) 'STATE VECTOR X?'
      CALL YESORN(ANSWER,DIARY)
      IF (ANSWER) GO TO 30
      IF(DIARY) THEN
         WRITE(4,*)' '
         WRITE(4,*)' '
         WRITE(4,*)' '
      END IF
      WRITE(*,*)' '
      WRITE(*,*)' '
      WRITE(*,*)' '
      RETURN
      END
C*********************************************************************|
      SUBROUTINE PNRDEF(PACH,P,R,NMAXPA,NMAXP,NMAXR,N,M,EFFECO,DIARY)
C                                                                     |
C     DEFINES MATRICES P AND R FROM PACH                              |
C*********************************************************************|
      REAL PACH(NMAXPA,N),P(NMAXP,N),R(NMAXR,N),TEST(15)
      LOGICAL DIARY
C
C  TESTING IF PACH(X,J) IS COMPLEX
C
      DO 10 J = 1, N
         TEST(J) = 0.0
  10  CONTINUE
      DO 20 J = 1, N
         DO 20 I = 1, N
            TEST(J) = TEST(J) + ABS(PACH(I+N,J))
  20  CONTINUE
C
C  DEFINING P
C
      DO 70 J = 1, N
```

```
            IF (TEST(J) .GE. 0.0) THEN
               DO 30 I = 1, N
                   P(I,J) = PACH(I,J)
   30          CONTINUE
               DO 40 I = 1, M
                   II = I+N+N
                   R(I,J) = PACH(II,J)
   40          CONTINUE
            ELSE
               JJ = J-1
               DO 50 I = 1, N
                   P(I,J) = PACH(I+N,JJ)
   50          CONTINUE
               DO 60 I = 1, M
                   R(I,J) = PACH(I+N+N+M,JJ)
   60          CONTINUE
            END IF
            IF (TEST(J) .GT. EFFEC0) TEST(J+1) = -TEST(J+1)
   70    CONTINUE
         IF(DIARY) THEN
            WRITE(4,*)' '
            WRITE(4,*)' '
            WRITE(4,*)' '
         END IF
         WRITE(*,*)' '
         WRITE(*,*)' '
         WRITE(*,*)' '
         RETURN
         END
C*************************************************************************|
         SUBROUTINE POLZER(A,B,N,M,DIARY)
C                                                                         |
C        FINDS THE POLES AND ZEROES IN THE FREQUENCY DOMAIN FOR THE       |
C        MATRIX A FROM THE TIME DOMAIN.  THIS WILL PRINT THE RESULTS.     |
C                                                                         |
C        USES SUBROUTINE RESOLV TO FIND THE RESOLVENT MATRIX, AND         |
C        SUBROUTINE PROOT TO FIND THE ROOTS OF THE POLYNOMIALS FOUND      |
C        BY RESOLV.                                                       |
C*************************************************************************|
         REAL A(15,15),Q(15,15,15),POLES(16),Z(16),V(15),U(15),B(15,15),
        +NUM(15,15,15)
         LOGICAL CHECK,DIARY
           IF(DIARY) THEN
              WRITE(4,*) 'DO YOU WANT TO CHECK THE RESOLVENT MATRIX'
              WRITE(4,*) '          ALGORITHM ACCURACY?'
           END IF
           WRITE(*,*) 'DO YOU WANT TO CHECK THE RESOLVENT MATRIX'
           WRITE(*,*) '          ALGORITHM ACCURACY?'
           CALL YESORN(CHECK,DIARY)
           CALL RESOLV(A,POLES,Q,N,CHECK,DIARY)
           POLES(N+1) = 1.0
           TEST = 0.0
           NTEST = N
           DO 10 K = N+1, 1, -1
```

```
         IF (POLES(K) .EQ. 0.0 .AND. TEST .EQ. 0.0) NTEST = I-2
         IF (POLES(K) .NE. 0.0) TEST = 1.0
10    CONTINUE
      IF(DIARY) THEN
      WRITE(4,*)' '
      WRITE(4,*)' '
      WRITE(4,*)'WHEN PRINTING THE COEFFICIENTS OF A POLYNOMIAL'
      WRITE(4,*)'GIVEN AS:'
      WRITE(4,*)' '
      WRITE(4,*)'A(1)X**N + A(2)X**(N-1) + ... + A(N)X + A(N+1)'
      WRITE(4,*)' '
      WRITE(4,*)'THE FIRST ELEMENT IS A(1), THE SECOND ELEMENT'
      WRITE(4,*)'IS A(2), AND SO ON.'
      WRITE(4,*)' '
      WRITE(4,*)' '
      WRITE(4,*)'THE COEFFICIENTS OF THE CHARACTERISTIC EQUATION:'
      WRITE(4,*)' '
      END IF
      WRITE(*,*)' '
      WRITE(*,*)' '
      WRITE(*,*)'WHEN PRINTING THE COEFFICIENTS OF A POLYNOMIAL'
      WRITE(*,*)'GIVEN AS:'
      WRITE(*,*)' '
      WRITE(*,*)'A(1)X**N + A(2)X**(N-1) + ... + A(N)X + A(N+1)'
      WRITE(*,*)' '
      WRITE(*,*)'THE FIRST ELEMENT IS A(1), THE SECOND ELEMENT'
      WRITE(*,*)'IS A(2), AND SO ON.'
      WRITE(*,*)' '
      WRITE(*,*)' '
      WRITE(*,*)'THE COEFFICIENTS OF THE CHARACTERISTIC EQUATION:'
      WRITE(*,*)' '
      DO 20 I = NTEST+1, 1, -1
         WRITE(*,*) POLES(I)
         IF(DIARY) WRITE(4,*) POLES(I)
20    CONTINUE
      IF(DIARY) THEN
         WRITE(4,*)' '
         WRITE(4,*)' '
         WRITE(4,*)'THE POLES:'
         WRITE(4,*)' '
      END IF
      WRITE(*,*)' '
      WRITE(*,*)' '
      CALL PROOT(NTEST,POLES,U,V,1)
      WRITE(*,*)'THE POLES:'
      WRITE(*,*)' '
      DO 30 I = 1, NTEST
         WRITE(*,*)U(I),'+',V(I),'i'
         IF(DIARY) WRITE(4,*)U(I),'+',V(I),'i'
30    CONTINUE
      DO 35 I = 1, N
         DO 35 J = 1, M
              DO 35 K = 1, N
                 NUM(I,J,K) = 0.0
```

```
35    CONTINUE
      DO 36 I = 1, N
         DO 36 J = 1, M
              DO 36 JJ = 1, N
                 DO 36 K = 1, N
                      NUM(I,J,K) = NUM(I,J,K) + Q(I,JJ,K) * B(JJ,J)
36    CONTINUE
      IF(DIARY) THEN
         WRITE(4,*)' '
         WRITE(4,*)' '
         WRITE(4,*)'THE POLYNOMIALS IN {ADJ(SI-A)}B ARE GIVEN AS:'
      END IF
      WRITE(*,*)' '
      WRITE(*,*)' '
      WRITE(*,*)'THE POLYNOMIALS IN {ADJ(SI-A)}B ARE GIVEN AS:'
      DO 80 I = 1, N
         DO 80 J = 1, M
            DO 40 K = 1, N
               Z(K) = NUM(I,J,K)
40          CONTINUE
            TEST = 0.0
            NTEST = N-1
            DO 50 K = N, 1, -1
               IF (Z(K) .EQ. 0.0 .AND. TEST .EQ. 0.0) NTEST = K-2
               IF (Z(K) .NE. 0.0) TEST = 1.0
50          CONTINUE
            WRITE(*,*)' '
            WRITE(*,*)' '
            WRITE(*,55) I,J
            IF(DIARY) THEN
               WRITE(4,*)' '
               WRITE(4,*)' '
               WRITE(4,55) I,J
            END IF
55          FORMAT(1X,'POLYNOMIAL IN POSITION(',I2,',',I2,')')
            IF (NTEST .LT. 0) THEN
              WRITE(*,*) '0.0'
              IF(DIARY) WRITE(4,*) '0.0'
            END IF
            DO 60 K = NTEST+1, 1, -1
               WRITE(*,*) Z(K)
               IF(DIARY) WRITE(4,*) Z(K)
60          CONTINUE
65          WRITE(*,*)' '
            WRITE(*,*)'THE ZEROES:'
            WRITE(*,*)' '
            IF(DIARY) THEN
               WRITE(4,*)' '
               WRITE(4,*)'THE ZEROES:'
               WRITE(4,*)' '
            END IF
            IF (NTEST .LT. 0) GO TO 80
               CALL PROOT(NTEST,Z,U,V,1)
               DO 70 K = 1, NTEST
```

```
                  WRITE(*,*) U(K),'+',V(K),'i'
                  IF(DIARY) WRITE(4,*) U(K),'+',V(K),'i'
 70             CONTINUE
 80    CONTINUE
       IF(DIARY) THEN
          WRITE(4,*) ' '
          WRITE(4,*) ' '
          WRITE(4,*) ' '
       END IF
       WRITE(*,*) ' '
       WRITE(*,*) ' '
       WRITE(*,*) ' '
       RETURN
       END
C*************************************************************************|
      SUBROUTINE PRINTA(NMAX,N,M,A,DIARY)
C                                                                         |
C     THIS SUBROUTINE PRINTS OUT AN ARBITRARY MATRIX A(N,M)               |
C*************************************************************************|
      REAL A(NMAX,M)
      LOGICAL DIARY
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
      END IF
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      DO 20 I=1,N
         WRITE(*,10) (A(I,J),J=1,M)
         WRITE(*,*) ' '
         IF(DIARY) THEN
            WRITE(4,10) (A(I,J),J=1,M)
            WRITE(4,*) ' '
         END IF
 10      FORMAT(5E14.5)
 20   CONTINUE
      IF(DIARY) THEN
         WRITE(4,*)' '
         WRITE(4,*)' '
         WRITE(4,*)' '
      END IF
      WRITE(*,*)' '
      WRITE(*,*)' '
      WRITE(*,*)' '
      RETURN
      END
C*************************************************************************|
      SUBROUTINE PRINTZ(Z,WR,WI,NMAX,N,DIARY)
C                                                                         |
C     PRINTS THE EIGENSTRUCTURE AS DEFINED BY Z, WI, AND WR               |
C*************************************************************************|
      REAL Z(NMAX,N),WR(NMAX),WI(NMAX)
      LOGICAL DIARY
      TEST = 0.0
```

```
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
      END IF
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      DO 10 J = 1, N
         TEST = TEST + ABS(WI(J))
10    CONTINUE
      IF(TEST .EQ. 0.0) THEN
         WRITE(*,*) 'THE VECTOR OF EIGENVALUES IS'
         IF(DIARY) WRITE(4,*) 'THE VECTOR OF EIGENVALUES IS'
         CALL PRNVEC(WR,NMAX,N,DIARY)
         IF(DIARY) THEN
            WRITE(4,*)' '
            WRITE(4,*)' '
            WRITE(4,*) 'THE MATRIX OF EIGENVECTORS IS'
         END IF
         WRITE(*,*)' '
         WRITE(*,*)' '
         WRITE(*,*) 'THE MATRIX OF EIGENVECTORS IS'
         CALL PRINTA(NMAX,N,N,Z,DIARY)
         IF(DIARY) THEN
            WRITE(4,*) 'THE J-TH COLUMN OF THIS MATRIX'
            WRITE(4,*) 'CORRESPONDS TO THE J-TH EIGENVALUE'
         END IF
         WRITE(*,*) 'THE J-TH COLUMN OF THIS MATRIX'
         WRITE(*,*) 'CORRESPONDS TO THE J-TH EIGENVALUE'
      ELSE
         CONJ = 1.0
         DO 50 J = 1, N
            IF(WI(J) .EQ. 0.0) THEN
                IF(DIARY) THEN
                   WRITE(4,*)' '
                   WRITE(4,*)' '
                   WRITE(4,15) J
                   WRITE(4,16) WR(J)
                   WRITE(4,*)' '
                   WRITE(4,*) 'AND THE CORRESPONDING EIGENVECTOR IS'
                   WRITE(4,*)' '
                END IF
                WRITE(*,*)' '
                WRITE(*,*)' '
                WRITE(*,15) J
15              FORMAT(1X,'EIGENVALUE NUMBER ',I2)
                WRITE(*,16) WR(J)
16              FORMAT(1X,'IS ',F15.8)
                WRITE(*,*)' '
                WRITE(*,*) 'AND THE CORRESPONDING EIGENVECTOR IS'
                WRITE(*,*)' '
                DO 20 I = 1, N
                   WRITE(*,*) Z(I,J)
                   IF(DIARY) WRITE(4,*) Z(I,J)
20              CONTINUE
```

```
                ELSE
                    IF(DIARY) THEN
                       WRITE(4,*)' '
                       WRITE(4,*)' '
                       WRITE(4,25) J
                       WRITE(4,*) 'IS',WR(J),' +',WI(J),'i'
                       WRITE(4,*)' '
                       WRITE(4,*) 'THE CORRESPONDING EIGENVECTOR IS'
                       WRITE(4,*)' '
                    END IF
                    WRITE(*,*)' '
                    WRITE(*,*)' '
                    WRITE(*,25) J
   25               FORMAT(1X,'EIGENVALUE NUMBER ',I2)
                    WRITE(*,*) 'IS',WR(J),' +',WI(J),'i'
                    WRITE(*,*)' '
                    WRITE(*,*) 'THE CORRESPONDING EIGENVECTOR IS'
                    WRITE(*,*)' '
                    IF(CONJ .GT. 0.0) THEN
                       DO 30 I = 1, N
                            WRITE(*,*) Z(I,J),'+',Z(I,J+1),'i'
                     IF(DIARY) WRITE(4,*) Z(I,J),'+',Z(I,J+1),'i'
   30                   CONTINUE
                       CONJ = -1.*CONJ
                    ELSE
                       CONJ = -1.*CONJ
                       DO 40 I = 1, N
                            ZI = -1.*Z(I,J)
                            WRITE(*,*) Z(I,J-1),'+',ZI,'i'
                       IF(DIARY) WRITE(4,*) Z(I,J-1),'+',ZI,'i'
   40                   CONTINUE
                    END IF
                END IF
   50        CONTINUE
             END IF
             IF(DIARY) THEN
                WRITE(4,*)' '
                WRITE(4,*)' '
                WRITE(4,*)' '
             END IF
             WRITE(*,*)' '
             WRITE(*,*)' '
             WRITE(*,*)' '
             RETURN
             END
C*************************************************************************|
             SUBROUTINE PRNVEC(S,NMAX,N,DIARY)
C                                                                         |
C            PRINTS AN ARBITRARY VECTOR S                                 |
C*************************************************************************|
             REAL S(NMAX)
             LOGICAL DIARY
             DO 10 I=1,N
                  WRITE(*,*) S(I)
```

```
            IF(DIARY) WRITE(4,*) S(I)
 10     CONTINUE
        RETURN
        END
C*************************************************************************
        SUBROUTINE PROOT(N,A,U,V,IR)
C
C       FINDS THE ROOTS OF A POLYNOMIAL WITH REAL COEFFICIENTS.
C       USES A MODIFIED BAIRSTOW METHOD.  ROUTINE FOUND IN "COMPUTER
C       PROGRAMS FOR COMPUTATIONAL ASSISTANCE IN THE STUDY OF LINEAR
C       CONTROL THEORY" BY MELSA AND JONES, MCGRAW-HILL BOOK COMPANY,
C       SECOND EDITION, 1973, PP 161-163.
C
C       N = DEGREE OF POLYNOMIAL, N < OR = 15.
C       A = POLYNOMIAL COEFFICIENT ARRAY.
C       U = REAL ROOT ARRAY.          V = IMAGINARY ROOT ARRAY.
C       IR = +1 IF POLYNOMIAL IS WRITTEN AS:
C              A(1) + A(2)S + ... + A(N+1)S**N
C       IR = -1 IF POLYNOMIAL IS WRITTEN AS:
C              A(1)S**N + A(2)S**(N-1) + ... + A(N+1).
C*************************************************************************
        DIMENSION A(16),U(16),V(16),H(17),B(17),C(17)
        IREV = IR
        NC = N + 1
        DO 10 I = 1, NC
 10        H(I) = A(I)
        P = 0.
        Q = 0.
        R = 0.
 20     IF(H(1)) 50,30,50
 30     NC = NC - 1
        V(NC) = 0.
        U(NC) = 0.
        DO 40 I = 1, NC
 40        H(I) = H(I+1)
        GO TO 20
 50     IF(NC-1) 60,480,60
 60     IF(NC-2) 80,70,80
 70     R = -H(1)/H(2)
        GO TO 320
 80     IF(NC-3) 100,90,100
 90     P = H(2)/H(3)
        Q = H(1)/H(3)
        GO TO 370
 100    IF(ABS(H(NC-1)/H(NC))-ABS(H(2)/H(1))) 110,170,170
 110    IREV = -IREV
        M = NC/2
        DO 120 I = 1, M
           NL = NC + 1 - I
           F = H(NL)
           H(NL) = H(I)
 120       H(I) = F
        IF(Q) 140,130,140
 130    P = 0.
```

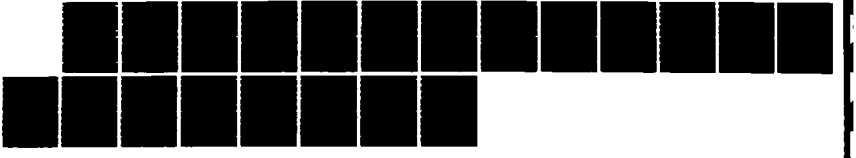
```
      GO TO 150
  140 P = P/Q
      Q = 1./Q
  150 IF(R) 160,170,160
  160 R = 1./R
  170 E = 5.E-10
      B(NC) = H(NC)
      C(NC) = H(NC)
      B(NC+1) = 0.
      C(NC+1) = 0.
      NP = NC-1
  180 DO 310 J = 1, 1000
         DO 190 I1 = 1, NP
            I = NC - I1
            B(I) = H(I) + R * B(I+1)
  190       C(I) = B(I) + R * C(I+1)
         IF(ABS(B(1)/H(1))-E) 320,320,200
  200    IF(C(2)) 220,210,220
  210    R = R + 1.
         GO TO 230
  220    R = R - B(1)/C(2)
  230    DO 240 I1 = 1, NP
            I = NC - I1
            B(I) = H(I) - P * B(I+1) - Q * B(I+2)
  240       C(I) = B(I) - P * C(I+1) - Q * C(I+2)
         IF(H(2)) 260,250,260
  250    IF(ABS(B(2)/H(1))-E) 270,270,280
  260    IF(ABS(B(2)/H(2))-E) 270,270,280
  270    IF(ABS(B(1)/H(1))-E) 370,370,280
  280    CBAR = C(2) - B(2)
         D = C(3)**2 - CBAR*C(4)
         IF(D) 300,290,300
  290    P = P - 2.
         Q = Q * (Q + 1.)
         GO TO 310
  300    P = P + (B(2) * C(3) - B(1) * C(4)) / D
         Q = Q + (-B(2) * CBAR + B(1) * C(3)) / D
  310 CONTINUE
      E = E * 10.
      GO TO 180
  320 NC = NC - 1
      V(NC) = 0.
      IF(IREV) 330,340,340
  330 U(NC) = 1. / R
      GO TO 350
  340 U(NC) = R
  350 DO 360 I = 1, NC
  360    H(I) = B(I+1)
      GO TO 50
  370 NC = NC - 2
      IF(IREV) 380,390,390
  380 QP = 1. / Q
      PP = P/(Q * 2.0)
      GO TO 400
```

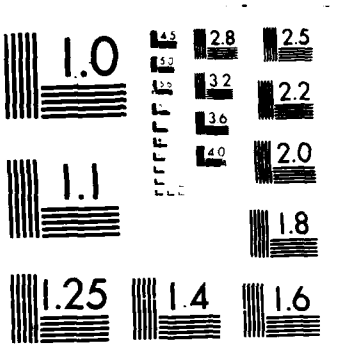
```
  390 QP = Q
      PP = P / 2.0
  400 F = (PP)**2 - QP
      IF(F) 410,420,420
  410 U(NC+1) = -PP
      U(NC) = -PP
      V(NC+1) = SQRT(-F)
      V(NC) = -V(NC+1)
      GO TO 460
  420 IF(PP) 440,430,440
  430 U(NC+1) = -SQRT(F)
      GO TO 450
  440 U(NC+1) = -(PP/ABS(PP)) * (ABS(PP) + SQRT(F))
  450 CONTINUE
      V(NC+1) = 0.
      U(NC) = QP / U(NC+1)
      V(NC) = 0.
  460 DO 470 I = 1, NC
  470    H(I) = B(I+2)
      GO TO 50
  480 RETURN
      END
C*********************************************************************
      SUBROUTINE RESOLV(AR,POLESR,QR,N,CHECK,DIARY)
C
C     A DOUBLE PRECISION ROUTINE TO CALCULATE THE RESOLVENT MATRIX
C
C     AR IS THE SINGLE PRECISION MATRIX OF INTEREST.
C     POLESR IS THE SINGLE PRECISION VECTOR OF THE COEFFICIENTS OF THE
C         CHARACTERISTIC EQUATION.
C     QR IS THE SINGLE PRECISION MATRIX OF POLYNOMIALS.
C     N IS THE ROW OR COLUMN DIMENSION OF THE N BY N A MATRIX.
C     CHECK IS A LOGICAL VARIBLE: CHECK = .TRUE. = PRINT CHECK OF
C                                                      ALGORITHM
C                                 CHECK = .FALSE. = NO PRINT OF CHECK
C*********************************************************************
      DOUBLE PRECISION A(15,15),Q(15,15,15),POLES(15),
     +Q1(15,15),Q2(15,15),TRACE,AIDENT(15,15)
      REAL AR(15,N),TEST(15,15),QR(15,15,15),POLESR(15)
      LOGICAL CHECK,DIARY
      N1 = N-1
      DO 10 I = 1, N
         DO 10 J = 1, N
            AIDENT(I,J) = 0.DO
            A(I,J) = DBLE(AR(I,J))
  10  CONTINUE
      DO 20 I = 1, N
         AIDENT(I,I) = 1.DO
  20  CONTINUE
      DO 30 I = 1, N
         DO 30 J = 1, N
            Q(I,J,N) = AIDENT(I,J)
  30  CONTINUE
      DO 40 I = 1, N
```

```
      DO 40 J = 1, N
         Q1(I,J) = Q(I,J,N)
40    CONTINUE
      CALL DAXBEC(Q1,15,N,N,A,15,N,N,Q2,15,DIARY)
      TRACE = 0.D0
      DO 50 I = 1, N
         TRACE = TRACE + Q2(I,I)
50    CONTINUE
      POLES(N) = -TRACE
      DIV = 1.D0
      DO 90 K = N1, 1, -1
         DIV = DIV + 1.D0
         TRACE = 0.D0
         DO 60 I = 1, N
            AIDENT(I,I) = POLES(K+1)
60       CONTINUE
         DO 70 I = 1, N
            DO 70 J = 1, N
               Q(I,J,K) = Q2(I,J) + AIDENT(I,J)
               Q1(I,J) = Q(I,J,K)
70       CONTINUE
         CALL DAXBEC(Q1,15,N,N,A,15,N,N,Q2,15,DIARY)
         DO 80 I = 1, N
            TRACE = TRACE + Q2(I,I)
80       CONTINUE
         POLES(K) = -TRACE/DIV
90    CONTINUE
      IF (CHECK) THEN
         DO 100 I = 1, N
            AIDENT(I,I) = POLES(1)
100      CONTINUE
         DO 110 I = 1, N
            DO 110 J = 1, N
               TEST(I,J) = REAL(Q2(I,J) + AIDENT(I,J))
110      CONTINUE
         IF(DIARY) THEN
         WRITE(4,*)'AS A CHECK,'
         WRITE(4,*)'THE FOLLOWING MATRIX SHOULD BE ALL ZEROES'
         END IF
         WRITE(*,*)'AS A CHECK,'
         WRITE(*,*)'THE FOLLOWING MATRIX SHOULD BE ALL ZEROES'
         CALL PRINTA(15,N,N,TEST,DIARY)
      END IF
      DO 130 K = N, 1, -1
         DO 120 I = 1, N
            DO 120 J = 1, N
               QR(I,J,K) = REAL(Q(I,J,K))
120      CONTINUE
         POLESR(K) = REAL(POLES(K))
130   CONTINUE
      RETURN
      END
```

AD-A179 341 MULTI-INPUT/MULTI-OUTPUT DESIGNATED EIGENSTRUCTURE (MODES): A COMPUTER-AI... (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI... M E HOPPER

UNCLASSIFIED MAR 87 AFIT/GAE/AA/87M-2 F/G 12/2 NL





```
C*********************************************************************|
      SUBROUTINE SAVE(A,B,C,N,M,DIARY,FILED,ITEST)
C                                                                     |
C     SAVES A MATRIX ON A COMPUTER STORAGE DEVICE                     |
C*********************************************************************|
      REAL A(15,M),B(30,M),C(30)
      CHARACTER FILEO*14,FILED*14,TEST*1
      LOGICAL OLD,DIARY
      WRITE(*,*)' '
      WRITE(*,*)' '
      WRITE(*,*)'WHAT IS THE OUTPUT FILE NAME?'
      IF(DIARY) THEN
         WRITE(4,*)' '
         WRITE(4,*)' '
         WRITE(4,*)'WHAT IS THE OUTPUT FILE NAME?'
         WRITE(*,*)' '
         WRITE(*,*)' '
      END IF
      READ(*,'(A)') FILEO
      IF(DIARY) THEN
         WRITE(4,*) FILEO
         WRITE(4,*)' '
         WRITE(4,*)' '
      END IF
      WRITE(*,*)' '
      WRITE(*,*)' '
   20 WRITE(*,30)
      IF(DIARY) WRITE(4,30)
   30 FORMAT(' IS OUTPUT FILE NEW OR OLD? ',//,' REPLY WITH: ',//,
     +' N  FOR NEW ',/,' O  FOR OLD ',/)
      READ(*,'(A1)') TEST
      IF(DIARY) WRITE(4,*) TEST
      OLD = LLE(TEST,'O') .AND. LGE(TEST,'O')
      IF (OLD) GO TO 40
      IF(LLT(TEST,'N') .OR. LGT(TEST,'N')) GO TO 20
      IF(DIARY) CLOSE(4)
      OPEN (4,FILE=FILEO,STATUS='NEW')
      GO TO 50
   40 IF(DIARY) CLOSE(4)
      OPEN (4,FILE=FILEO,STATUS='UNKNOWN')
   50 CONTINUE
      DO 60 I = 1, N
      IF(ITEST .EQ. 1) WRITE(4,*) (A(I,J),J=1,M)
      IF(ITEST .EQ. 2) WRITE(4,*) (B(I,J),J=1,M)
      IF(ITEST .EQ. 3) WRITE(4,*) C(I)
   60 CONTINUE
      CLOSE(4)
      IF(DIARY) THEN
        OPEN(4,FILE=FILED,STATUS='NEW')
        WRITE(*,*) ' '
        WRITE(*,*) ' '
        WRITE(*,*) ' '
      END IF
      WRITE(*,*) ' '
```

```
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      RETURN
      END
C*********************************************************************|
      SUBROUTINE SETUPC(A,B,EVALR,EVALI,SETUP,N,M,NX2)
C                                                                     |
C     SETUPS THE WORKING MATRIX WHEN ANY ONE EIGENVALUE IS IMAGINARY  |
C*********************************************************************|
      REAL A(15,N),B(15,M),SETUPT(30,60),LAMBRE(15,15),
     +LAMBIM(15,15),SETUP(60,NX2)
      DO 20 I = 1, N
         DO 10 J = 1, N
            LAMBRE(I,J) = 0.0
            LAMBIM(I,J) = 0.0
  10     CONTINUE
         LAMBRE(I,I) = EVALR
         LAMBIM (I,I) = EVALI
  20  CONTINUE
      DO 30 I = 1, N
         DO 30 J = 1, N
            SETUPT(I,J) = A(I,J) - LAMBRE(I,J)
            SETUPT(I,J+N) = LAMBIM(I,J)
            SETUPT(I+N,J) = -1.*LAMBIM(I,J)
            SETUPT(I+N,J+N) = SETUPT(I,J)
  30  CONTINUE
      DO 40 I = 1, N
         DO 40 J = 1, M
            SETUPT(I,2*N+J) = B(I,J)
            SETUPT(I,2*N+M+J) = 0.0
            SETUPT(N+I,2*N+J) = 0.0
            SETUPT(N+I,2*N+M+J) = B(I,J)
  40  CONTINUE
      CALL ATRANS(30,60,2*N,2*(N+M),SETUPT,SETUP)
      RETURN
      END
C*********************************************************************|
      SUBROUTINE SIGPL(S,SIGPLS,NMAXS,NMAXSP,N,M,EFFECO)
C                                                                     |
C     DEFINES THE REQUIRED MATRIX SIGMA-PLUS FROM THE SINGULAR VALUES |
C*********************************************************************|
      REAL S(NMAXS),SIGPLS(NMAXSP,N)
      INTEGER R
      DO 10 I = 1, N
         IF(S(I) .GT. EFFECO) R = I
  10  CONTINUE
      DO 20 I = 1, M
         DO 20 J = 1, N
            SIGPLS(I,J) = 0.0
  20  CONTINUE
      DO 30 I = 1, R
         SIGPLS(I,I) = 1./S(I)
  30  CONTINUE
      RETURN
```

```
      END
C*********************************************************************|
      SUBROUTINE SIMUL8(A,B,N,M,NMAX,DIARY)
C                                                                     |
C     A AND B ARE THE MATRICES FOR THE EQUATION X(DOT) = AX + BU      |
C     FOR WHICH WE WANT A PLOT OF THE TIME HISTORY.                   |
C                                                                     |
C     A IS DIMENSIONED N X N    B IS DIMENSIONED N X M (M = # INPUTS).|
C     NM IS THE ROW DIMENSION OF BOTH A AND B AS FOUND IN THE MAIN    |
C     PROGRAM.  ASUBT AND BSUBT ARE THE MATRICES GENERATED FOR A      |
C     DISCRETE TIME SIMULATION FOR A GENERAL MIMO SYSTEM AS FOUND     |
C     IN "LINEAR SYSTEM FUNDAMENTALS" BY J. GARY REID, PAGE 273.      |
C                                                                     |
C*********************************************************************|
      REAL A(NMAX,N),B(NMAX,M),ASUBT(15,15),BSUBT(15,15),E(15,15),
     +T,ETEMP(15,15),XHISTR(15,151),XI(15),XII(15),USUBT(15),
     +IDENT(15,15)
      LOGICAL ANSWER,DIARY
  1   WRITE(*,*) ' '
      WRITE(*,*) 'ENTER THE TIME INCREMENT FOR'
      WRITE(*,*) 'THE TIME DISCRETIZATION.'
      WRITE(*,*) ' '
      READ(*,*) T
      WRITE(*,*) ' '
      WRITE(*,*) 'ENTER THE FINAL TIME FOR THE SIMULATION.'
      READ(*,*) TF
      IF(DIARY) THEN
      WRITE(4,*) ' '
      WRITE(4,*) 'ENTER THE TIME INCREMENT FOR'
      WRITE(4,*) 'THE TIME DISCRETIZATION.'
      WRITE(4,*) ' '
      WRITE(4,*) '  USE A DECIMAL POINT WHEN ENTERING THE INCREMENT.'
      WRITE(4,*) T
      WRITE(4,*) ' '
      WRITE(4,*) 'ENTER THE FINAL TIME FOR THE SIMULATION.'
      WRITE(4,*) TF
      END IF
      K = NINT(TF/T)
      IF(K .GT. 150) THEN
         IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
         WRITE(4,*) 'THIS CHOICE OF TIME INCREMENT AND FINAL TIME'
         WRITE(4,*) 'REQUIRES TOO MANY POINTS.  SELECT AGAIN SUCH'
         WRITE(4,*) 'THAT FINAL TIME / TIME INCREMENT < 150'
         END IF
         WRITE(*,*) ' '
         WRITE(*,*) ' '
         WRITE(*,*) 'THIS CHOICE OF TIME INCREMENT AND FINAL TIME'
         WRITE(*,*) 'REQUIRES TOO MANY POINTS.  SELECT AGAIN SUCH'
         WRITE(*,*) 'THAT FINAL TIME / TIME INCREMENT < 150'
         GO TO 1
      END IF
      TOL = 1.E-8
```

```
      DIV = 2.0
      AF = 0.0
      DO 5 I = 1, M
         IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,2) I
         WRITE(4,*) 'FOR A STEP INPUT WITH ZERO INITIAL CONDITIONS'
         END IF
         WRITE(*,*) ' '
         WRITE(*,2) I
    2    FORMAT(1X,'ENTER THE VALUE OF INPUT VARIABLE # ',I2)
         WRITE(*,*) 'FOR A STEP INPUT WITH ZERO INITIAL CONDITIONS'
         READ(*,*) USUBT(I)
         IF(DIARY) WRITE(4,*) USUBT(I)
    5 CONTINUE
      DO 10 I = 1, N
         XI(I) = 0.0
   10 CONTINUE
      DO 15 I = 1,N
         DO 15 J = 1,N
            AF = A(I,J)*A(I,J) + AF
   15 CONTINUE
      AF = SQRT(AF)
C
C     AF IS THE FROEBINIUS NORM OF THE MATRIX A.
C     SEE PAGE 276 OF "LINEAR SYSTEM FUNDAMENTALS"
C
C     NOW TO FIND AN APPROPRIATE SQUARING COEFFICIENT NN FOR THE
C     SQUARING ALGORITHM
C
      TEST = AF*T/2.0
      NN = 1
   20 IF(TEST.LT.1.0) GO TO 30
      TEST = TEST/2.0
      NN = NN + 1
      GO TO 20
   30 QUO = T/(2.0**NN)
C
C     DEFINING THE IDENTITY MATRIX AND FIRST TERM OF THE
C     E-SUB-0 POWER SERIES
C
      DO 50 I = 1,N
         DO 40 J = 1,N
            IDENT(I,J) = 0.0
            E(I,J) = 0.0
   40    CONTINUE
         IDENT(I,I) = 1.0
         E(I,I) = QUO
   50 CONTINUE
C
C     DEFINING THE SECOND TERM OF POWER SERIES - TO BE USED FOR EACH
C     SUCCESSIVE TERM, AND ADDS IT TO E-SUB-0
C
      DO 60 I = 1,N
```

```
            DO 60 J = 1,N
               BSUBT(I,J) = A(I,J)*QUO*QUO/2.0
   60    CONTINUE
   70    DO 80 I = 1,N
            DO 80 J = 1,N
               E(I,J) = E(I,J) + BSUBT(I,J)
   80    CONTINUE
C
C        HERE, BSUBT IS THE LAST ELEMENT OF THE POWER SERIES
C
         DIV = DIV + 1.0
         CALL AXBEQC(A,NMAX,N,N,BSUBT,15,N,N,ETEMP,15,DIARY)
         HIGH = 0.0
         DO 90 I = 1,N
            DO 90 J = 1,N
               ETEMP(I,J) = ETEMP(I,J)*QUO/DIV
               TEST = ABS(BSUBT(I,J) - ETEMP(I,J))
               HIGH = AMAX1(TEST,HIGH)
               BSUBT(I,J) = ETEMP(I,J)
   90    CONTINUE
         IF(HIGH.GT.TOL) GO TO 70
         DO 100 I = 1,N
            DO 100 J = 1,N
               E(I,J) = E(I,J) + BSUBT(I,J)
  100    CONTINUE
C
C        THE MATRIX E-SUB-0 IS NOW KNOWN AS E(I,J).
C        NOW TO DEFINE F-SUB-0 (ASUBT(I,J)).
C
         CALL AXBEQC(A,NMAX,N,N,E,15,N,N,ASUBT,15,DIARY)
         DO 110 I = 1,N
            DO 110 J = 1,N
               ASUBT(I,J) = ASUBT(I,J) + IDENT(I,J)
  110    CONTINUE
C
C        CALCULATING E-SUB-N (E(I,J)) AND F-SUB-N
C
         DO 130 MM = 1,NN
            CALL AXBEQC(ASUBT,15,N,N,E,15,N,N,BSUBT,15,DIARY)
            DO 120 I = 1,N
               DO 120 J = 1,N
                  E(I,J) = E(I,J) + BSUBT(I,J)
                  BSUBT(I,J) = ASUBT(I,J)
  120       CONTINUE
            CALL AXBEQC(BSUBT,15,N,N,BSUBT,15,N,N,ASUBT,15,DIARY)
  130    CONTINUE
C
C        THIS ASUBT IS THE SAME A-SUB-T USED IN "LINEAR SYSTEM
C        FUNDAMENTALS"
C
         CALL AXBEQC(E,15,N,N,B,NMAX,N,M,BSUBT,15,DIARY)
C
C        LIKEWISE, THIS IS THE DESIRED B-SUB-T.
C        NOW TO CALCULATE THE TIME HISTORY OF X.
```

```
C
      DO 180 KOUNT = 1, K+1
         IF (KOUNT .EQ. 1) GO TO 150
         CALL AXBEQC(ASUBT,15,N,N,XI,15,N,1,XII,15,DIARY)
         CALL AXBEQC(BSUBT,15,N,M,USUBT,15,M,1,XI,15,DIARY)
         DO 140 I = 1,N
            XI(I) = XI(I) + XII(I)
 140     CONTINUE
 150     DO 170 I = 1,N
            XHISTR(I,KOUNT) = XI(I)
 170     CONTINUE
 180  CONTINUE
      KP1 = K + 1
      CALL PLOT(XHISTR,KP1,N,T,NMAX,DIARY)
      IF(DIARY) THEN
      WRITE(4,*) ' '
      WRITE(4,*) ' '
      WRITE(4,*) ' '
      WRITE(4,*) 'DO YOU WANT TO RE-DO THE TIME SIMULATION WITH OTHER'
      WRITE(4,*) 'TIME PARAMETERS?'
      END IF
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) 'DO YOU WANT TO RE-DO THE TIME SIMULATION WITH OTHER'
      WRITE(*,*) 'TIME PARAMETERS?'
      CALL YESORN(ANSWER,DIARY)
      IF(ANSWER) GO TO 1
      IF(DIARY) THEN
         WRITE(4,*) ' '
         WRITE(4,*) ' '
         WRITE(4,*) ' '
      END IF
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      WRITE(*,*) ' '
      RETURN
      END
C*********************************************************************|
      SUBROUTINE SUBST(W,B,X,IPIVOT,N,NMAX)
C                                                                     |
C     USED TO INVERT A MATRIX WITH SUBROUTINE FACTOR                  |
C*********************************************************************|
      DIMENSION W(NMAX,N),B(N),X(N),IPIVOT(N)
      IF (N .GT. 1) GO TO 10
      X(1) = B(1) / W(1,1)
      RETURN
 10   IP = IPIVOT(1)
      X(1) = B(IP)
      DO 15 K = 2, N
         IP = IPIVOT(K)
         KMI = K - 1
         SUM = 0.
         DO 14 J = 1, KMI
```

```
            SUM = W(IP,J) * X(J) + SUM
 14       CONTINUE
          X(K) = B(IP) - SUM
 15    CONTINUE
       X(N) = X(N) / W(IP,N)
       K = N
       DO 20 NPIMK = 2, N
          KPI = K
          K = K - 1
          IP = IPIVOT(K)
          SUM = 0.
          DO 19 J = KPI, N
             SUM = W(IP,J) * X(J) + SUM
 19       CONTINUE
          X(K) = (X(K) - SUM) / W(IP,K)
 20    CONTINUE
       RETURN
       END
C*************************************************************************|
       SUBROUTINE SVD(A,MMAX,NMAX,M,N,NU,NV,S,U,V,IER)
C                                                                         |
C      USES A MODIFIED ACM ROUTINE CSVD.                                  |
C*************************************************************************|
       REAL A(MMAX,N),U(MMAX,M),V(NMAX,N),S(N),B(60),C(60),T(60),
     +Q,R
       INTEGER P
       DATA ETA,TOL/1.5E-8,1.E-31/
       P=0
       NP=N+P
       N1=N+1
C
C      HOUSEHOLDER REDUCTION
       C(1)=0.E0
       K=1
  10   K1=K+1
C
C      ELIMINATION OF A(I,K), I=K+1,...,M
       Z=0.E0
       DO 20 I=K,M
  20      Z=Z+A(I,K)**2
       B(K)=0.E0
       IF(Z.LE.TOL)GOTO 70
       Z=SQRT(Z)
       B(K)=Z
       W=ABS(A(K,K))
       Q=1.E0
       IF(W.NE.0.E0)Q=A(K,K)/W
       A(K,K)=Q*(Z+W)
       IF(K.EQ.NP)GOTO 70
       DO 50 J=K1,NP
          Q=0.E0
          DO 30 I=K,M
  30         Q=Q+A(I,K)*A(I,J)
          Q=Q/(Z*(Z+W))
```

```
            DO 40 I=K,M
  40           A(I,J)=A(I,J)-Q*A(I,K)
  50        CONTINUE
C
C   PHASE TRANSFORMATION
          Q=-A(K,K)/ABS(A(K,K))
          DO 60 J=K1,NP
  60        A(K,J)=Q*A(K,J)
C
C    ELIMINATION OF A(K,J), J=K+2,...,N
  70    IF(K.EQ.N)GOTO 140
        Z=0.E0
        DO 80 J=K1,N
  80        Z=Z+A(K,J)**2
        C(K1)=0.E0
        IF(Z.LE.TOL)GOTO 130
        Z=SQRT(Z)
        C(K1)=Z
        W=ABS(A(K,K1))
        Q=1.E0
        IF(W.NE.0.E0)Q=A(K,K1)/W
        A(K,K1)=Q*(Z+W)
        DO 110 I=K1,M
           Q=0.E0
           DO 90 J=K1,N
  90          Q=Q+A(K,J)*A(I,J)
           Q=Q/(Z*(Z+W))
           DO 100 J=K1,N
 100          A(I,J)=A(I,J)-Q*A(K,J)
 110       CONTINUE
C
C   PHASE TRANSFORMATION
          Q=-A(K,K1)/ABS(A(K,K1))
          DO 120 I=K1,M
 120        A(I,K1)=A(I,K1)*Q
 130    K=K1
        GOTO 10
C
C TOLERANCE FOR NEGLIGIBLE ELEMENTS
 140    EPS=0.E0
        DO 150 K=1,N
           S(K)=B(K)
           T(K)=C(K)
 150       EPS=AMAX1(EPS,S(K)+T(K))
        EPS=EPS*ETA
C
C INITIALIZATION OF U AND V
        IF(NU.EQ.0)GOTO 180
        DO 170 J=1,NU
           DO 160 I=1,M
 160          U(I,J)=0.E0
 170       U(J,J)=1.E0
 180  IF(NV.EQ.0)GOTO 210
      DO 200 J=1,NV
```

```
      DO 190 I=1,N
 190     V(I,J)=0.E0
 200  V(J,J)=1.E0
C
C QR DIAGONALIZATION
 210  DO 380 KK=1,N
         K=N1-KK
         ITS=0
C
C  TEST FOR SPLIT
 220     DO 230 LL=1,K
            L=K+1-LL
            IF(ABS(T(L)).LE.EPS)GOTO 290
            IF(ABS(S(L-1)).LE.EPS)GOTO 240
 230   CONTINUE
C
C CANCELLATION OF E(L)
 240   CS=0.E0
       SN=1.E0
       L1=L-1
       DO 280 I=L,K
          F=SN*T(I)
          T(I)=CS*T(I)
          IF(ABS(F).LE.EPS)GOTO 290
          H=S(I)
          W=SQRT(F*F+H*H)
          S(I)=W
          CS=H/W
          SN=-F/W
          IF(NU.EQ.0)GOTO 260
          DO 250 J=1,N
             X=U(J,L1)
             Y=U(J,I)
             U(J,L1)=X*CS+Y*SN
 250         U(J,I)=Y*CS-X*SN
 260      IF(NP.EQ.N)GOTO 280
          DO 270 J=N1,NP
             Q=A(L1,J)
             R=A(I,J)
             A(L1,J)=Q*CS+R*SN
 270         A(I,J)=R*CS-Q*SN
 280      CONTINUE
C
C    TEST FOR CONVERGENCE
 290      W=S(K)
          IF(L.EQ.K)GOTO 360
          IF(ITS.EQ.30) GOTO 565
          ITS=ITS+1
C
C   ORIGIN SHIFT
          X=S(L)
          Y=S(K-1)
          G=T(K-1)
          H=T(K)
```

```
            F=((Y-W)*(Y+W)+(G-H)*(G+H))/(2.E0*H*Y)
            G=SQRT(F*F+1.E0)
            IF(F.LT.0.E0)G=-G
            F=((X-W)*(X+W)+(Y/(F+G)-H)*H)/X
C
C  QR STEP
      CS=1.E0
      SN=1.E0
      L1=L+1
      DO 350 I=L1,K
         G=T(I)
         Y=S(I)
         H=SN*G
         G=CS*G
         W=SQRT(H*H+F*F)
         T(I-1)=W
         CS=F/W
         SN=H/W
         F=X*CS+G*SN
         G=G*CS-X*SN
         H=Y*SN
         Y=Y*CS
         IF(NV.EQ.0)GOTO 310
         DO 300 J=1,N
            X=V(J,I-1)
            W=V(J,I)
            V(J,I-1)=X*CS+W*SN
300         V(J,I)=W*CS-X*SN
310      W=SQRT(H*H+F*F)
         S(I-1)=W
         CS=F/W
         SN=H/W
         F=CS*G+SN*Y
         X=CS*Y-SN*G
         IF(NU.EQ.0)GOTO 330
         DO 320 J=1,N
            Y=U(J,I-1)
            W=U(J,I)
            U(J,I-1)=Y*CS+W*SN
320         U(J,I)=W*CS-Y*SN
330      IF(N.EQ.NP)GOTO 350
         DO 340 J=N1,NP
            Q=A(I-1,J)
            R=A(I,J)
            A(I-1,J)=Q*CS*R*SN
340         A(I,J)=R*CS-Q*SN
350      CONTINUE
         T(L)=0.E0
         T(K)=F
         S(K)=X
         GOTO 220
C
C   CONVERGENCE
360      IF(W.GE.0.E0)GOTO 380
```

```
            S(K)=-W
            IF(NV.EQ.0)GOTO 380
            DO 370 J=1,N
  370          V(J,K)=-V(J,K)
  380       CONTINUE
C
C     SORT SINGULAR VALUES
          DO 450 K=1,N
             G=-1.E0
             J=K
             DO 390 I=K,N
                IF(S(I).LE.G)GOTO 390
                G=S(I)
                J=I
  390           CONTINUE
             IF(J.EQ.K)GOTO 450
             S(J)=S(K)
             S(K)=G
             IF(NV.EQ.0)GOTO 410
             DO 400 I=1,N
                Q=V(I,J)
                V(I,J)=V(I,K)
  400           V(I,K)=Q
  410        IF(NU.EQ.0)GOTO 430
             DO 420 I=1,N
                Q=U(I,J)
                U(I,J)=U(I,K)
  420           U(I,K)=Q
  430        IF(N.EQ.NP)GOTO 450
             DO 440 I=N1,NP
                Q=A(J,I)
                A(J,I)=A(K,I)
  440           A(K,I)=Q
  450        CONTINUE
C
C     BACK TRANSFORMATION
          IF(NU.EQ.0)GOTO 510
          DO 500 KK=1,N
             K=N1-KK
             IF(B(K).EQ.0.E0)GOTO 500
             Q=-A(K,K)/ABS(A(K,K))
             DO 460 J=1,NU
  460           U(K,J)=Q*U(K,J)
             DO 490 J=1,NU
                Q=0.E0
                DO 470 I=K,M
  470              Q=Q+A(I,K)*U(I,J)
                Q=Q/(ABS(A(K,K))*B(K))
                DO 480 I=K,M
  480              U(I,J)=U(I,J)-Q*A(I,K)
  490           CONTINUE
  500        CONTINUE
  510     IF(NV.EQ.0)GOTO 570
          IF(N.LT.2)GOTO 570
```

```
      DO 560 KK=2,N
         K=N1-KK
         K1=K+1
         IF(C(K1).EQ.0.E0)GOTO 560
         Q=-A(K,K1)/ABS(A(K,K1))
         DO 520 J=1,NV
520         V(K1,J)=Q*V(K1,J)
         DO 550 J=1,NV
            Q=0.E0
            DO 530 I=K1,N
530            Q=Q+A(K,I)*V(I,J)
            Q=Q/(ABS(A(K,K1))*C(K1))
            DO 540 I=K1,N
540            V(I,J)=V(I,J)-Q*A(K,I)
550         CONTINUE
560      CONTINUE
      GOTO 570
C     SET ERROR - NO CONVERGENCE TO A SINGULAR VALUE
C     AFTER 30 ITERATIONS
565   IER=9999
570   RETURN
      END
C*********************************************************************|
      SUBROUTINE V2DEF(V,NMV,N,S,NMS,V2,NV2,EFFECO,DIARY)
C                                                                     |
C   DETERMINES THE NULL SPACE GIVEN V AND THE ORDERED SINGULAR VALUES |
C*********************************************************************|
      REAL S(NMS),EFFECO,V(NMV,N),V2(NMV,N)
      LOGICAL DIARY
      DO 10 I = 1, N
         IF(S(I) .GT. EFFECO) II = I
10    CONTINUE
      NV2 = N - II
      IF(NV2 .EQ. 0) THEN
         WRITE(*,*) 'THE MATRIX IS OF FULL RANK'
         IF(DIARY) WRITE(4,*) 'THE MATRIX IS OF FULL RANK'
      ELSE
          DO 20 I = 1, N
              DO 20 J = 1, NV2
                   V2(I,J) = V(I,J+II)
20        CONTINUE
      END IF
      RETURN
      END
C*********************************************************************|
      SUBROUTINE V2SDEF(V2,V2S,NV2MAX,NV2SMX,N,M)
C                                                                     |
C     DEFINES THE REQUIRED SUB-MATRIX V2S FROM V2                     |
C*********************************************************************|
      REAL V2(NV2MAX,M),V2S(NV2SMX,M)
      DO 10 I = 1, N
         DO 10 J = 1, M
            V2S(I,J) = V2(I,J)
 10   CONTINUE
```

```
      RETURN
      END
C*********************************************************************|
      SUBROUTINE YESORN(ANSWER,DIARY)
C                                                                     |
C     FINDS THE ANSWER TO THE QUESTION ASKED IN THE CALLING PROGRAM   |
C*********************************************************************|
      CHARACTER*1 TEST
      LOGICAL ANSWER,DIARY
 10   WRITE(*,*) '          ENTER Y FOR YES OR N FOR NO'
      IF(DIARY) WRITE(4,*) '          ENTER Y FOR YES OR N FOR NO'
      READ(*,'(A1)') TEST
      IF(DIARY) WRITE(4,*) TEST
      ANSWER = ((LLE(TEST,'Y') .AND. LGE(TEST,'Y')) .OR.
     +(LLE(TEST,'y') .AND. LGE(TEST,'y')))
      IF (ANSWER) GO TO 20
      IF((LLT(TEST,'N') .OR. LGT(TEST,'N')) .AND.
     +(LLT(TEST,'n') .OR. LGT(TEST,'n'))) GO TO 10
 20   IF(DIARY) WRITE(4,*) ' '
      WRITE(*,*) ' '
      RETURN
      END
```

Bibliography

Alag, Gurbux S., and Eugene L. Duke. "Development of Control Laws for a Flight Test Maneuver Autopilot," Journal of Guidance, Vol. 9, No. 4: 441-445 (July-August 1986).

Businger, Peter A., and Gene H. Golub. "Algorithm 358: Singular Decomposition of a Complex Matrix," Collected Algorithms From ACM, Vol. 2, Book 1. New York: The Association for Computing Machinery, Inc., 1980.

Elbert, Theodore F. Estimation and Control of Systems. New York: Von Nostrand and Reinhold Company, 1984.

Gantmacher, F.R. The Theory of Matrices, Vol. 1. New York: Chelsea Publishing Company, 1959.

Golub, G., and W. Kahan. "Calculating the Singular Values and Pseudo-Inverse of a Matrix," Journal of the Society of Industrial Applied Mathematics, Numerical Analysis, Ser. B, Vol.2, No. 2: 205-224 (1965).

Golub, G.H., and C. Reinsch. "Singular Value Decomposition and Least Squares Solutions," Numerische Mathematik, 14: 403-420 (1970).

Hamming, R.W. Numerical Methods for Scientists and Engineers. New York: McGraw-Hill Book Company, Inc. 1962.

Johnson, Lee W., and R. Dean Reiss. Numerical Analysis (Second Edition). Reading, Massachusetts: Addison-Wesley Publishing Company, 1982.

Klema, Virginia C., and Alan J Laub. "The Singular Value Decomposition: Its Computation and Some Applications," IEEE Transactions on Automatic Control, AC-25 (2): 164-176 (April 1980).

Martin, R.S., and J.H. Wilkinson. "Similarity Reduction of a General Matrix to Hessenberg Form," Numerische Mathematik, 12: 349-368 (December 1968).

Melsa, James L., and Stephen K. Jones. Computer Programs for Computational Assistance in the Study of Linear Control Theory (Second Edition). New York: McGraw-Hill Book Company, 1973.

Parlett, B.N., and C. Reinsch. "Balancing a Matrix for Calculation of Eigenvalues and Eigenvectors," Numerische Mathematik, 13: 293-304 (August 1969).

Peters, G., and J.H. Wilkinson. "Eigenvectors of Real and Complex Matrices by LR and QR Triangularizations," Numerische Mathematik, 16: 181-204 (1970).

Press, William H., et al. Numerical Recipes The Art of Scientific Computing. New York: Cambridge University Press, 1986.

Ralston, Anthony, and Philip Rabinowitz. A First Course in Numerical Analysis (Second Edition). New York: McGraw-Hill Book Company, 1978.

Reid, J. Gary. Linear System Fundamentals Continuous and Discrete, Classic and Modern. New York: McGraw-Hill Book Company, 1983.

Silverthorn, James T., and J. Gary Reid. "Computation of the Subspaces for Entire Eigenstructure Assignment Via the Singular Value Decomposition," Proceedings of the 19th IEEE Conference on Decision and Control, Vol.2: 1206, 1207 (December 1980).

Smith, B.T., et al. Matrix Eigensystem Routines-EISPACK Guide. Lecture Notes in Computer Science, Vol. 6 (Second Edition), edited by G. Goos and J. Hartmanis. New York: Springer-Verlag, 1976.

Sobel, K.M., and Shapiro, E.Y. "A Design Methodology for Pitch Pointing Flight Control Systems," Journal of Guidance, Vol. 8, No. 2: 181-187 (March-April 1985).

Spang, H. Austin, III. "The Federated Computer-Aided Control Design System," Proceedings of the IEEE, 72 (12): 1724-1731 (December 1984).

Strang, Gilbert. Linear Algebra and Its Applications. New York: Academic Press, Inc., 1976.

Uspensky, J.V. Theory of Equations. New York: McGraw-Hill Book Company, 1948.

Walker, Robert A., et al. "Computer-Aided Engineering (CAE) for System Analysis," Proceedings of the IEEE, 72 (12): 1732-1745 (December 1984).

Williams, T.W.C. "Numerically Reliable Software for Control: The SLICE Library," IEEE Proceedings, 133 (2): 73-82 (March 1986).

VITA

Michael Eric Hopper was born on June 3, 1960 in Amarillo, Texas. He graduated from high school in Manilao, Guam in 1978. In 1982, he graduated with honors from the University of Oklahoma with the degree of Bachelor of Science, Aerospace Engineering. During his senior year, he worked as a product support engineer for the Israel Aircraft Industries' Westwind corporate jet aircraft. As an ROTC Distinguished Graduate, he received his Regular commission shortly after he entered the U.S. Air Force. Michael Hopper started working on his master's degree as a part-time student while working as the lead engineer for aerodynamics, flight testing, configuration control, and first flight certification for the X-29 Program Office of the Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio. He completed his master's degree while working as the Test and Evaluation Manager for the Advanced Tactical Fighter System Program Office, Wright-Patterson AFB, Ohio.

Permanent address: 11600 Ashford Drive
Yukon, Oklahoma 73099

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GAE/AA/87M-2	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENY	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
(U) Multi-Input/Multi-output Designated Eigenstructure (MODES): A Computer-Aided Control System Design Program

12. PERSONAL AUTHOR(S)
Michael E. Hopper, Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1987, March	210

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Control Systems, Flight Control Systems, Eigenvalues, Eigenvectors, Optimization, Control Theory
01	04		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Daniel Gleason, Thesis Advisor

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Daniel Gleason, Instructor, AFIT/ENY	22b TELEPHONE (Include Area Code) (513) 255-2362 22c OFFICE SYMBOL AFIT/ENY

The design engineer can specify the closed-loop response of a system, including both the modes and mode shapes, with the eigenstructure assignment approach to system synthesis. This effort was to create an interactive computer-aided control system design program to enable the designer to use the modern control approach to design a control system with specified response characteristics, or to come as close to the ideal as possible. With the program MODES, the design engineer can easily calculate the feedback gains needed to give the controlled system the optimum response characteristics by specifying the desired eigenvalues and eigenvectors. The singular value decomposi-position is used to accomplish this.

In addition, MODES can provide valuable information about the controlled system to assist the designer in specifying the desired eigenstructure. By calculating the eigenvalues and eigenvectors of the original system, by plotting the time response history of the original system, and by calculating the resolvent matrix, poles and zeroes for classical control analysis, MODES is a strong tool in determining how the eigenstructure should be altered. By presenting the same information for the modified "optimized" system, MODES can show where additional changes to the specified desired eigenstructure should be made.

The accuracy of MODES is verified with sample applications found in the technical literature.

END

5-87

DTIC